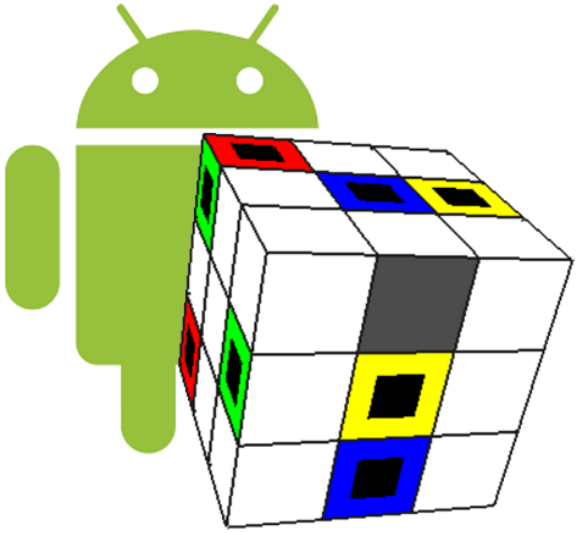
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Videojoc 3D per Android: 3D Logic

Projecte de Final de Carrera

Xavier Torremadé Barreda
12/01/2012



Director: Lluís Pérez Vidal
Departament: Llenguatges i Sistemes Informàtics (LSI)
Titulació: Enginyeria en Informàtica

Taula de continguts

1 Introducció

1.1 Descripció

El Projecte de Final de Carrera és una adaptació del joc 3D Logic[1] per a dispositius Android. Es disposa d'un cub, on cada cara està formada per MxN quadrats petits. En el global del cub hi ha inicialment marcats de 1 a n parells de quadrats (cada parell pintat amb el mateix color). L'objectiu és crear un camí entre cada parella de quadrats pintant els quadrats lliures del cub amb el mateix color.

La complicació resideix en poder comunicar tots els parells de quadrats, doncs no es pot acolorir un quadrat amb dos colors diferents i hi ha quadrats que no estan disponibles per acolorir (quadrats bloquejats). El joc consta de diferents nivells, augmentant gradualment la dificultat.

Es farà servir un servidor extern per guardar-hi estadístiques de la resolució dels diferents nivells pels usuaris. L'aplicació mòbil es connectarà a aquest servidor per consultar classificacions i les mostrarà a l'usuari.

El joc està desenvolupat amb les llibreries especifiques d'Android i en Java. S'ha fet servir com a IDE Eclipse i per la part gràfica els frameworks que adapten OpenGL en aquests dispositius.

1.2 Motivació

Aquest projecte em permet cobrir dos de les grans motivacions personals que tinc en la meva vida.

La primera és la meva **afició als videojocs**. Des de petit m'he sentit atret per aquest mon, per jocs de tots els gèneres, duracions i temàtiques. Aquesta afició la segueixo tenint, tot i que molt evolucionada respecte a la meva infantesa/adolescència, amb un punt de vista més crític: l'edat fa que et sorprenguis cada cop menys. Però tot i així, m'agrada estar dins d'aquest món, motiu pel qual col·laboro a la revista online de videojocs Mundogamers.com com a editor i coordinador de continguts.

La segona motivació és la de tot allò que envolta **Internet i dispositius mòbils**. Considero que Internet ens va *"conectar al món"*, i que els smartphones ens estan *"conectant al món EN TOT MOMENT"*. I em sembla fascinant aquest fet.

A més a més, durant la carrera vaig fer una assignatura de lliure elecció anomenada: "Taller de programació d'aplicacions Android per a Google Phones" on vaig aprendre les bases de la programació per Android, i on vam fer varis projectes entre els quals destacaria el "Guide & Vote", un projecte de localització geogràfica

[1] Joc per ordinador desenvolupat per Alex Matveev i dissenyat per Poll Harvey

de punts d'interès en ciutats. I també m'he començat a endinsar en el món iOS per desenvolupar per iPhone/Android, provant alguns frameworks com Cocos2D, per a crear jocs en 2D.

Per tant, agraeixo molt que la universitat permeti fer aquest tipus de projecte. He sofert per part d'algunes persones certa animadversió o sorpresa quan els he explicat la temàtica del meu projecte: els hi semblava molt estrany que un videojoc per Android pogués ser un projecte de final de carrera.

Però a mi em sembla just el contrari. Considero que si la UPC permet presentar aquest tipus de projectes vol dir que **la universitat s'apropa a la realitat que ens envolta**: tant el desenvolupament de jocs, com les aplicacions per dispositius mòbils son dos dels mercats més productius del sector de les TIC.

I no només això, sinó que també considero que, pel futur professional dels enginyers, **el desenvolupament d'aplicacions mòbils es una molt bona via per emprendre** (juntament amb el mon de les xarxes socials), degut principalment als baixos costos inicials que té començar un projecte d'aquest tipus respecte d'altres negocis més tradicionals.

1.3 Objectiu

L'objectiu proposat des d'un principi és que l'usuari pogués jugar al cub de 3D Logic, és a dir, pintar quadrats, rotar el cub, i que l'aplicació se n'adoni de quan l'usuari ha trobat la solució del problema.

També l'aplicació havia de tenir diferents nivells, augmentant-ne gradualment la dificultat, i certes característiques pròpies de gairebé la totalitat de jocs Android d'avui en dia, com poder configurar el so (encès/apagat) o guardar el nombre de nivells completats per l'usuari.

Durant la realització del projecte vaig creure convenient afegir algun objectiu més al projecte, i es va decidir dotar al joc d'un sistema d'estadístiques centralitzat.

1.4 Organització de la memòria

La memòria està organitzada en els següents capítols:

El primer capítol és en el que ens trobem, la introducció, on he repassat la descripció del projecte, he descrit la motivació i els objectius i on estic fent un breu resum de les seccions més importants de la memòria.

El segon capítol es l'anàlisi del projecte. El *què*, i no el *com*. Tractaré dels requisits funcionals i els requisits no funcionals del projecte i faré una breu justificació de certes decisions que afectaran més tard al disseny del projecte.

El tercer capítol és el disseny del projecte, l'apartat més important i per tant més extens de la memòria. Aquí es descriurà *com* ha sigut desenvolupat 3D Logic, entrant en la profunditat necessària però sense entrar mai detalls d'implementació no necessaris per a entendre el funcionament global de l'aplicació. Per tant es farà servir esquemes conceptuals i diagrames de seqüència en UML, pseudocodi i gràfics explicatius.

El quart capítol son els mapes navegacionals. Aquest capítol té l'objectiu d'ensenyar com son les interaccions entre les diferents pantalles de l'aplicació, així com descriure els elements (per exemple botons) que hi ha en pantalla.

El cinquè capítol parlarà de la planificació del projecte. Per a fer-ho s'exposarà un diagrama de Gantt explicant les tasques que s'han realitzat i quant s'ha tardat per cada una, així com les desviacions que han anat apareixent durant la realització del projecte.

El sisè capítol tractarà sobre les possibles millores que podria tenir el projecte. 3D Logic podria ser ampliat de moltes maneres i no totes aquestes millores podien fer-se amb el nombre d'hores dedicat a un Projecte de Final de Carrera, però no per això no deixen de ser interessants com a millores de futur.

El setè capítol seran les conclusions que he extret de realitzar aquest projecte. La meva opinió sobre la plataforma de desenvolupament (Android+OpenGL), problemes que he tingut al dur-lo a terme, etc.

El vuitè capítol serà un annex explicant el disseny del servidor d'estadístiques: servei web i base de dades.

I finalment el novè i últim capítol serà la bibliografia del projecte.

2 Anàlisis

2.1 Requeriments funcionals

A continuació explicarem els requeriments funcionals del projecte, és a dir, el comportament del software, sense entrar en el disseny o la implementació del projecte.

2.1.1 Nivell de joc

Estem davant del requeriment més important del projecte. Al cap i a la fi, l'usuari el que voldrà es jugar al joc. Per tant, no només li hem de mostrar correctament a l'usuari el nivell de joc, sinó que també li hem de donar les eines necessàries per a que el pugui jugar correctament.

Per tant, els elements indispensables que li hem de oferir a l'usuari son:

- Visualització del cub.
- Rotació del cub
- Selecció d'un quadrat.

Però a més a més, a l'usuari se li oferiran altres elements per tal de que l'experiència de joc sigui més complerta:

- Cartell mostrant el nivell del joc en el que ens trobem.
- Marcador amb la puntuació del nivell.
- Botó esborrat, per esborrar tots els quadrats pintats per l'usuari (a mode de *reset*).

2.1.2 Emmagatzematge de les preferències de l'usuari

El joc ha de guardar les preferències de la partida de l'usuari. En el nostre cas guardarem dos informacions: el nombre de nivells ja completats i si el so està activat o no.

Sobre guardar la informació del nombre de nivells completats, ens servirà per poder bloquejar els nivells que no estiguin completats, excepte el primer, per tal de que l'usuari hagi de seguir l'ordre dels nivells. Aquesta informació s'ha de persistir per a que l'usuari pugui entrar i sortir de l'aplicació quan vulgui i no hagi de tornar a fer aquells nivells que ja ha completat.

Respecte el so, es guardarà la informació de l'estat del so, és a dir, si el so està encès o apagat. Estem davant d'una aplicació mòbil i és important que aquesta s'adapti a l'usuari. Si l'usuari, en l'última partida que ha fet al joc ha decidit desactivar-lo, el més probable estadísticament és que quan torni a jugar el vulgui desactivat també.

Aquesta informació quedarà persistent al mòbil i només s'eliminarà quan l'usuari decideixi desinstal·lar l'aplicació.

2.1.3 Gestió d'estadístiques

Un cop l'usuari estigui jugant a l'aplicació, i en concret a un nivell qualsevol del joc, se li mostrarà en pantalla un marcador amb la puntuació. Aquest marcador es calcularà en funció del temps destinat a resoldre el nivell: com més temps, menys puntuació.

Aquestes puntuacions s'enregistraran, si l'usuari vol i si disposa de connexió a internet, en un sistema d'estadístiques centralitzat. D'aquesta manera, qualsevol usuari de 3D Logic pot introduir el seu nom i puntuació de cada un dels nivells que jugui.

A més a més, l'aplicació disposarà de pantalles per consultar les estadístiques. Aquestes es podran consultar a partir del nivell del joc o del nom de l'usuari.

2.2 Requeriments no funcionals

2.2.1 Eficiència

Tot i que en l'actualitat hi ha dispositius Android molt potents (per exemple el Samsung Galaxy Nexus o el HTC Sensation), els dispositius mòbils tradicionalment han sigut de recursos limitats. Evidentment, quan es desenvolupa per Android l'objectiu és que aquesta aplicació es pugui executar correctament en tots o en la gran majoria dels seus dispositius.

A més a més, 3D Logic es una aplicació que fa ús de gràfics. És important, doncs, que no es realitzin càlculs complexos dins d'aquelles tasques que es realitzen freqüentment (mètodes de draw/render). També s'intentarà, sempre que sigui possible, guardar en memòria aquella informació que es pugui reaprofitar per tal de no recalcular-la parcial o totalment en els mètodes esmentats anteriorment.

2.2.2 Usabilitat

Al programar en dispositius Android s'ha de tenir en compte que es treballa amb pantalles petites i de baixa resolució. Per tant, un aspecte molt important a tenir en compte al desenvolupar 3D Logic és que les pantalles del joc continguin únicament la informació essencial.

Per aquest motiu, cada pantalla tindrà ben definida la seva funcionalitat, i si se'n requereix d'alguna més es crearà una nova pantalla i es navegarà en ella per tenir les pantalles el més simples possibles.

2.2.3 Portabilitat

La portabilitat en Android s'entén com la capacitat del programa per a funcionar correctament en tots els dispositius i versions d'Android. Per funcionar correctament s'entén no només que el programa no produeixi errors, sinó també que els seus elements es vegin de forma adequada: per exemple, que els botons surtin tots dins de la pantalla o que no es solapin entre ells.

Per desgràcia, aquest és un dels més grans talons d'Aquiles del desenvolupament d'aplicacions Android, sobretot si ho comparem amb la competència més directa (iOS d'Apple).

Els desenvolupadors d'Android disposem d'un emulador amb el que provar l'aplicació amb diferents configuracions. El problema és que aquest emulador no és fidel al 100% i el seu rendiment es molt baix, la qual cosa dificulta les proves.

Per tant, he hagut d'acotar els tests i aquests s'han realitzat en els dispositius Samsung Galaxy S[2] (també la plataforma de desenvolupament) i HTC Sensation[3], comprovant la correcta visualització de 3D Logic en diferents versions d'Android i mides de pantalla.

2.3 Decisions de disseny

2.3.1 Cares del cub de 3x3 quadrats

Degut a que treballem sobre pantalles petites i de baixa resolució, el cub tindrà les cares de 3x3 quadrats. Fer més quadrats per cares implicaria que l'usuari s'equivocaria al clicar els quadrats amb molta més freqüència.

Una altra opció hagués sigut implementar un zoom i així permetre cares de 4x4 o fins i tot 5x5 quadrats, però llavors el desenvolupament de cada nivell es faria molt més tediós per l'usuari.

Per tant, s'ha decidit que les cares seran de 3x3 quadrats i que totes les cares del cub poden estar disponibles i s'aniran activant a mesura que l'usuari avanci, és a dir, a mesura que la dificultat del joc vagi augmentant. Per exemple, en els últims nivells, disposarem de 6 cares i 9 quadrats per cara, un total de 54 quadrats.

---

[2] Pantalla de 4", amb resolució de 800x400 píxels i versió 2.2 Froyo.
[3] Pantalla de 4.3", amb resolució de 960x540 píxels i versió 2.3.3 Gingerbread.

2.3.2 Versió d'OpenGL ES 1.0

OpenGL ES (OpenGL for Embedded Systems) és una variant simplificada de l'API d'OpenGL dissenyada per dispositius com mòbils o PDAs, entre elles els sistemes Android. Actualment hi ha 3 versions: OpenGL ES 1.0 (basada en OpenGL 1.3), OpenGL ES 1.1 (basada en OpenGL 1.5) i OpenGL ES 2.0 (basada en OpenGL 2.0).

Per 3D Logic s'ha triat la versió 1.0, seguint els criteris d'elecció que marca Android Developers[4]:

- **Rendiment:** OpenGL ES 2.0 es mostra més ràpid que OpenGL ES 1.0/1.1, però depèn dels dispositius i de les diferències de la implementació de la pipeline d'OpenGL. En el cas de 3D Logic, no s'aprofitaria la potència de la versió 2.0 i no hi hauria diferències de rendiment significatives.

- **Compatibilitat amb dispositius:** OpenGL ES 1.0/1.1 son suportats des d'Android 1.0, mentre que la versió 2.0 és suportada pels dispositius Android 2.2 i superiors. En aquest punt és millor OpenGL ES 1.0/1.1, tot i que ja en l'actualitat només un 10% dels dispositius Android en el mercat tenen una versió inferior a 2.2[5].

- **Codificació:** OpenGL ES 1.0/1.1 disposa d'un pipeline fix i una sèrie de funcions que fa que sigui molt més fàcil i ràpid programar que per la versió 2.0. Per exemple, la versió 2.0 obliga l'ús de shaders, els quals afegirien un grau de complexitat al projecte.

- **Control de gràfics:** OpenGL ES 2.0 permet un control més acurat del pipeline al ser totalment programable fent ús de shaders, la qual cosa permet generar efectes que amb 1.0/1.1 serien molt difícils de fer. Tot i així, per 3D Logic no és necessària aquesta potència.

[4] http://developer.android.com/guide/topics/graphics/opengl.html#choosing-version

[5] http://developer.android.com/resources/dashboard/platform-versions.html

3 Disseny

En aquest capítol s'explicarà el procés de disseny de l'aplicació mòbil, amb la màxima claredat possible, detallant els aspectes més rellevants del projecte i justificant les decisions preses.

Per assegurar la claredat en l'explicació, no s'han detallat tots els aspectes del projecte. No es volia entrar en nivells d'implementació que ens allunyarien del fil principal. Però, si es volen conèixer aquests detalls, es pot consultar el codi font adjuntat.

Per consultar com ha estat dissenyat la part servidor, consultar el capítol ANNEX: Disseny servidor.

3.1 Arquitectura del sistema

L'arquitectura del projecte és la següent: disposem d'un dispositiu mòbil amb el joc i un sistema d'estadístiques centralitzat en un servidor. Tenim un servidor d'aplicacions GlassFish que rep les peticions dels diferents dispositius mòbils i en guarda/consulta la informació en una petita base de dades MySQL.

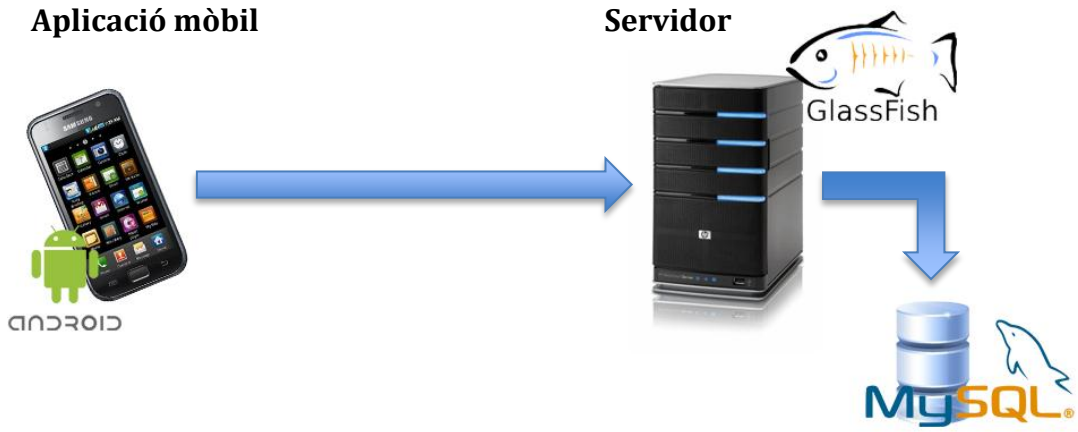


Figura 3.1. Arquitectura del sistema.

L'aplicació mòbil s'ha dividit en 3 paquets que diferencien les 3 parts més importants de l'aplicació: el paquets Android, OpenGL i WebService[6].

[6] Quan en aquest apartat parlo del paquet WebService, em refereixo a la part client, és a dir, a tot aquella lògica necessària per fer ús dels serveis web des de l'aplicació mòbil.

3.2 Paquet Android

A continuació descriurem la "part Android" de l'aplicació, és a dir, tots aquells components pròpiament d'Android. Això inclou tota la navegació entre pantalles, gestió de preferències de l'usuari i visualització d'estadístiques.

3.2.1 Consideracions sobre Android

Abans de començar, volia fer una petita introducció per tenir clars certs conceptes que son imprescindibles per entendre la part Android de 3D Logic.

Òbviament, no és l'objectiu d'aquesta memòria explicar tot el sistema Android, per això ja hi ha la pàgina oficial d'Android Developers amb tota la informació i molts exemples, però sí que amb un capítol es poden donar les 3 pinzellades necessàries per entendre el que ve a continuació.

Activity

Durant les següents pàgines parlarem d'una classe anomenada Activity. Aquesta classe s'encarrega de la lògica de les pantalles de les aplicacions Android. Típicament, per crear pantalles en una aplicació estendrem d'aquesta classe[7] i sobreescriurem els mètodes que creguem convenients i que gestionen l'anomenat Activity Lifecycle, o sigui, el cicle de vida de les Activity.

No entrarem en detalls sobre el cicle de vida però sí que explicarem els mètodes que sobreescriurem a 3D Logic:

- **onCreate:** On s'inicialitza l'Activity. A més a més, és on es defineix la vista de la interfície d'usuari.

- **onClick:** Gestiona els events de click per part de l'usuari.

- **onResume:** Es crida quan l'Activity començarà a interactuar amb l'usuari.

- **onActivityResult:** Es crida un cop l'Activity llençada per una Activity finalitza.

- **onBackPressed:** Es crida quan es detecta que l'usuari ha premut el botó d'enrere del dispositiu.

[7] En 3DLogic estendrem de la classe LogicActivity, que estén de la classe Activity, com veurem més endavant.

Comunicació entre Activities: Intents

Per comunicar-nos entre dos Activity per, per exemple, canviar de pantalla, tenim a la nostra disposició dos mètodes:

- **startActivity:** Per començar una Activity nova.

- **startActivityForResult:** Es comença una segona Activity i, un cop aquesta finalitza amb la crida finish(), es torna a la primera Activity cridant el mètode onActivityResult. A més a més, aquest mètode pot rebre un resultat creat en la segona Activity, per a que la primera el tracti.

Ambdós mètodes disposen d'un mecanisme per comunicar-se, els Intent, és a dir, que la primera Activity pot enviar informació[8] a la segona Activity mitjançant aquesta classe.

Interfície d'usuari

Com hem comentat anteriorment, al mètode onCreate de l'Activity es defineix la interfície d'usuari. Aquesta es defineix mitjançant la crida setContentView referenciant un arxiu XML que conté els elements típics per representar pantalles, com per exemple:

- **Layouts:** Per situar els elements en pantalla.

- **Buttons:** Botons per a que l'usuari cliqui.

- **TextBoxs:** Per a que l'usuari introdueixi dades.

- **TextViews:** Mostren informació a l'usuari no modificable.

Tots aquests elements son subclasse de View, que és el bloc bàsic per representar components d'interfície d'usuari. Una View ocupa una àrea rectangular i ens permet pintar i gestionar *events* d'usuari.

En 3D Logic, cada Activity tindrà el seu layout, amb el mateix nom, al directori res/layout.

[8] Es poden enviar tipus bàsics (enters, booleans, bytes, strings...) i Bundles, que permet posar-hi tot tipus de dades.

3.2.2 Model conceptual (UML)

A continuació mostrem el diagrama conceptual de la "part Android" i en els següents apartats se'n descriuran els detalls més rellevants:

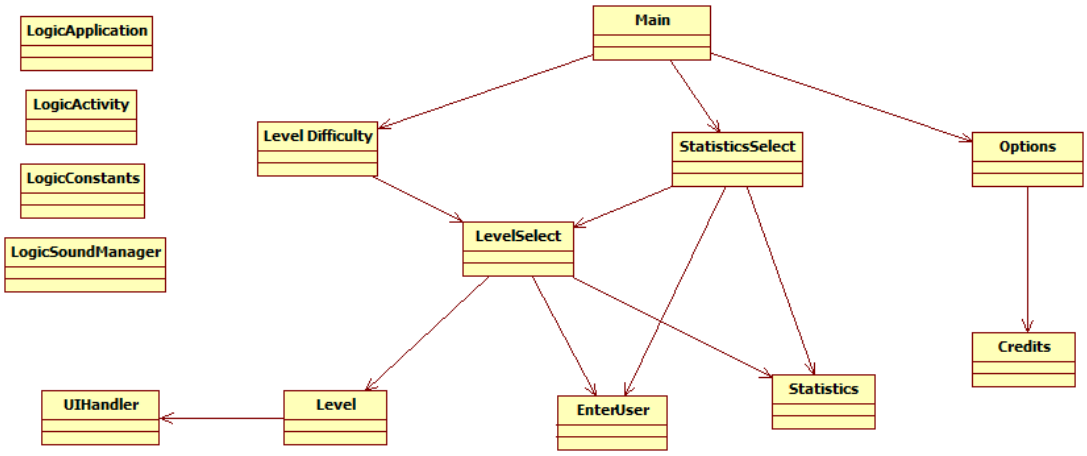


Figura 3.2. Diagrama UML amb les interaccions entre Activities.

3.2.3 LogicAplication

Tal i com indica Android Developers, la classe Application[9] es la classe bàsica per mantenir un estat global en l'aplicació.

Per a 3D Logic, estendrem[10] aquesta classe amb la LogicApplication, que ens servirà per guardar l'estat global de l'usuari. Si l'usuari ha estat introduït ja a l'aplicació per jugar, aquest quedarà guardat i no tornarà a ser preguntat fins que l'usuari faci un *reset* d'usuari (dins la pantalla opcions).

[9] http://developer.android.com/reference/android/app/Application.html

[10] Si bé es cert que no faria falta fer una subclasse de Application i es podria fer simplement amb una singleton, considerava més elegant fer servir la mateixa Application, que a més es pot obtenir fàcilment mitjançant les típiques crides d'Android.

3.2.4 LogicActivity

Una bona forma per no repetir codi entre les diferents Activity que formen part d'una aplicació Android es posar tot el codi comú en una classe que estengui la Activity i fer que la resta d'Activities de l'aplicació estenguin d'aquesta nova classe.

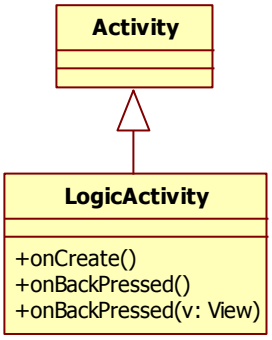


Figura 3.3. Especificació de la classe LogicActivity mostrant els mètodes que sobreescriurem.

En 3D Logic, aquesta classe es la LogicActivity, que proporcionarà funcionalitats de so comuns a totes les Activities:

- Al mètode onCreate, **configuració del so per tal de que al prémer els botons físic de volum del mòbil es pugi el so multimèdia** i no el so del to de trucada. Així, es podrà pujar i baixar el so del joc mentre s'està jugant.

- **Personalització del mètode onBackPressed**[11]. Aquest mètode és cridat quan l'usuari prem el botó físic d'enrere, o també amb el botó d'enrere del propi joc. Farem que al prémer el botó es reprodueixi el so corresponent.

3.2.5 LogicSoundManager

La gestió dels sons en Android es pot fer de dos maneres:

- **MediaPlayer:** Dissenyat per arxius de so llargs. S'ajusta més per arxius de música, per exemple per una música de fons executant-se durant l'aplicació. Els arxius reproduïts amb MediaPlayer son carregats de disc, salvant espai en memòria però introduint un petit retràs no apreciable.

---

[11] Com es pot observar a l'UML, hi ha dos *onBackPressed* públics definits. Això és perquè *onBackPressed()* és el mètode que sobreescrivim d'Activity, cridat quan es prem el botó físic d'enrere; i *onBackPressed(v: View)* és el mètode públic cridat quan es prem el botó Android d'enrere situat a la part inferior esquerre de la pantalla, que es defineix al layout XML de les pantalles, amb la propietat *onClick*, que necessàriament envia el paràmetre View a la funció que li referenciem.

- **SoundPool:** Dissenyat per petits clips de so. Aquests poden ser guardats en memòria i descomprimits ràpidament, per tant és la classe perfecta per aquells efectes de so curts de l'aplicació. No s'han de fer servir per sons o música gran, degut a que poden excedir la memòria màxima.

Per 3D Logic, degut a que només tenim efectes de so i no una música de fons, farem servir la classe SoundPool. Per aquest objectiu, s'ha creat una classe anomenada LogicSoundManager que disposa de mètodes per carregar i reproduir els sons que es desitgi.

3.2.6 Shared Preferences

Android dona diversos mètodes per guardar i persistir informació de l'aplicació[12]. **Per 3D Logic es farà servir Shared Preferences.**

Aquesta classe permet guardar dades primitives (booleans, ints, floats, longs i strings) en **format clau-valor**. A més a més, permet tenir diferents arxius de preferències, identificats per un nom, tot i que per a 3D Logic només ens en farà falta un únic arxiu.

Per tant, en aquesta classe hi guardarem:

- **L'últim nivell completat per l'usuari.**

- **L'estat del so (activat/desactivat)**, per a que quan l'usuari torni a jugar al joc, aquest tingui el so configurat tal i com estava en l'última vegada.

3.2.7 UIHandler

Com hem explicat anteriorment, LogicRenderer estén el Renderer de GLSurfaceView per dur a terme tota la visualització en OpenGL.

Tots els procediments d'OpenGL es realitzen en un nou thread. Per tant, aquells elements que han estat creats en el thread d'Android (el thread de UI, User Interface), no poden ésser modificats en el thread OpenGL sense un sistema de comunicació entre threads.

Aquí es on entra en joc la classe UIHandler. Un Handler[13] en Android és una classe que permet enviar i processar missatges associats a una cua de missatges d'un thread. Cada instància d'un Handler està associada a un sol thread i a una sola cua de missatges de threads.

Per tant, nosaltres creem un UIHandler (que estén Handler) durant la inicialització del nivell, estant encara en el thread UI. Aquest UIHandler és enviat a

[12] http://developer.android.com/guide/topics/data/data-storage.html
[13] http://developer.android.com/reference/android/os/Handler.html

LogicRenderer per a que pugui ser utilitzat. Quan necessitem enviar un *event* i retornar l'execució al thread UI, simplement des de LogicRenderer enviem un missatge a UIHandler:

Aquest sistema ens permetrà, per exemple, poder escriure missatges en pantalla de tipus Toast durant l'execució del nivell, per a mostrar el nivell en el que estem o per mostrar un missatge d'error en cas de que no es puguin guardar les estadístiques per un error de connexió.

3.2.8 Cas d'ús: Jugar nivell

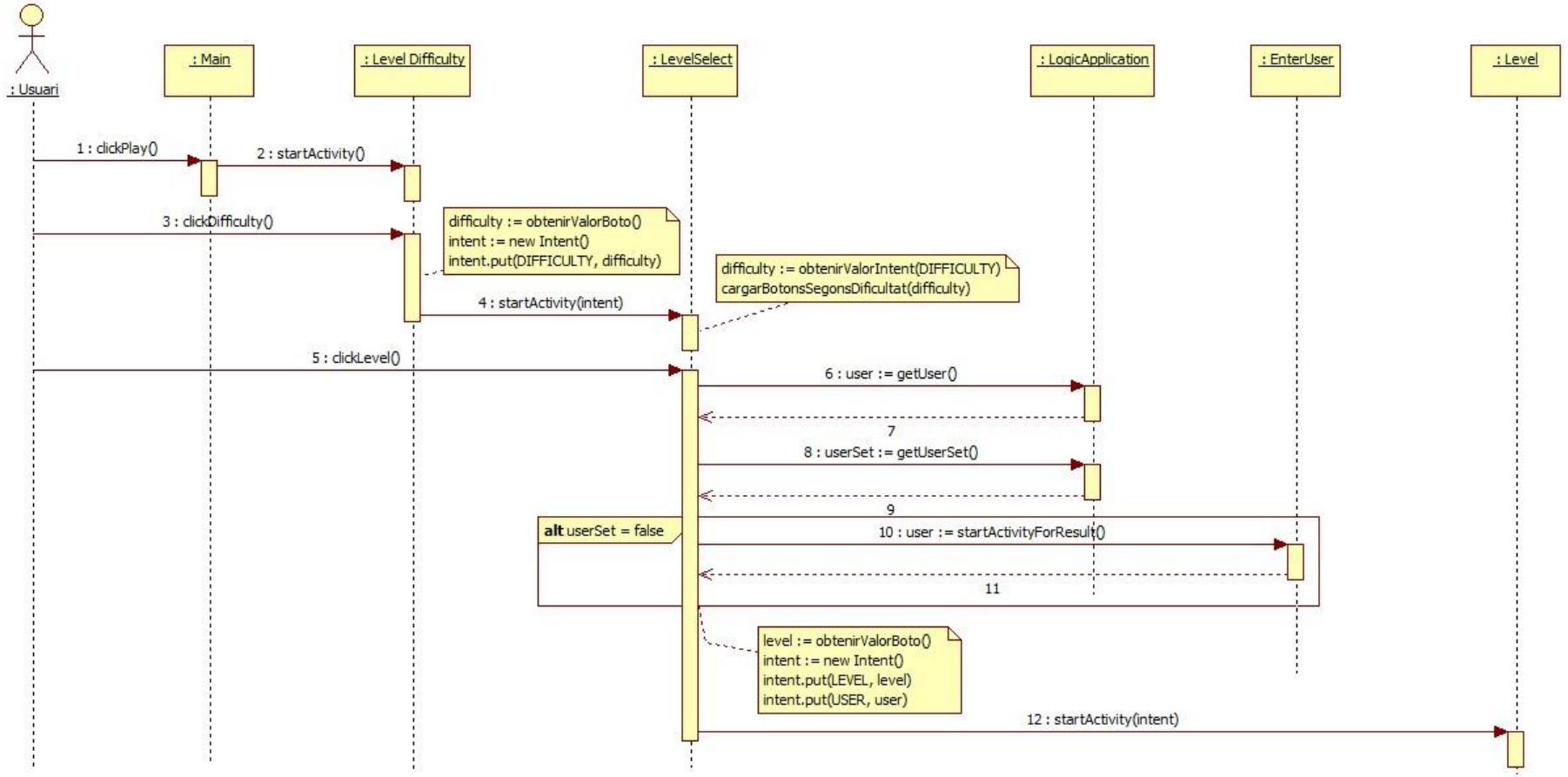


Figura 3.4. Diagrama de seqüència del cas d'ús: Jugar nivell.

L'usuari clica el botó de play, fet que provoca que la Activity principal **Main** comenci l'Activity **LevelDifficulty**.

En aquesta pantalla, l'usuari pot triar entre 3 dificultats: EASY, MEDIUM o HARD, i un cop fet s'iniciarà la Activity **LevelSelect**, enviant-li un Intent indicant quin nivell de dificultat ha triat l'usuari.

A **LevelSelect**, s'obtindrà l'Intent i la informació del nivell de dificultat i es carregaran els botons de nivell. Els botons de nivell estan definits al layout XML de **LevelDificulty**, concretament la mida del botó, el fons i el seu comportament genèric. El que farà la Activity és indicar a aquests botons si han d'estar activats o no i quin text ha de mostrar: si venim de EASY els botons seran del 1 al 9, MEDIUM del 10 al 18 i HARD del 19 al 27.

Quan l'usuari cliqui un dels botons, l'Activity comprovarà si l'usuari ja ha introduït el seu nom d'usuari (pel sistema d'estadístiques) anteriorment, fent les crides necessàries a la classe **LogicApplication**.

En cas de que no ho hagi fet, s'iniciarà amb *startActivityForResult* l'Activity **EnterUser** i l'usuari podrà introduir l'usuari o decidir jugar offline. Un cop ho hagi fet, aquesta Activity finalitzarà i retornarem a **LevelSelect**, on obtindrem la informació de l'usuari introduït.

Tant si rebem la informació d'usuari de l'Activity **EnterUser** com si ja estava introduïda, LevelSelect prepararà l'intent amb tota la informació necessària per començar l'Activity **Level**, que posteriorment donarà el control al thread OpenGL i realitzarà el pintat del nivell.

3.2.9 Cas d'ús: Consultar estadístiques de nivell

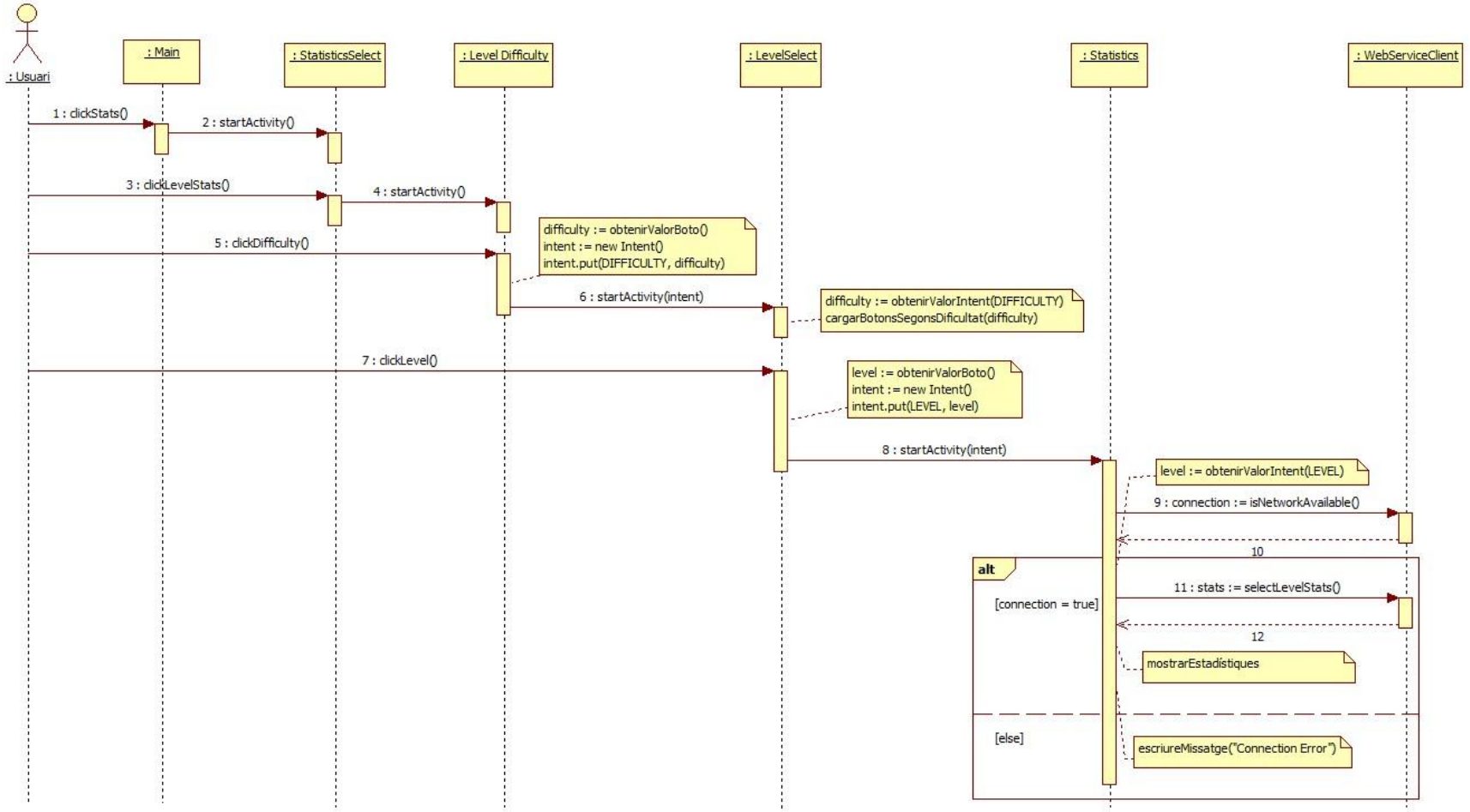


Figura 3.5. Diagrama de seqüència del cas d'ús: Consultar estadístiques de nivell.

L'usuari clica el botó de stats, fet que provoca que la Activity principal **Main** comenci l'Activity **StatisticsSelect**. Aquí, l'usuari clicarà el botó d'estadístiques de nivell i s'iniciarà l'Activity **LevelDifficulty**.

En aquesta pantalla, l'usuari pot triar entre 3 dificultats: EASY, MEDIUM o HARD, i un cop fet s'iniciarà la Activity **LevelSelect**, enviant-li un Intent indicant quin nivell de dificultat ha triat l'usuari.

A **LevelSelect**, s'obtindrà l'Intent i la informació del nivell de dificultat i es carregaran els botons de nivell. Els botons de nivell estan definits al layout XML de **LevelDificulty**, concretament la mida del botó, el fons i el seu comportament genèric. El que farà la Activity és indicar a aquests botons quin text ha de mostrar: si venim de EASY els botons seran del 1 al 9, MEDIUM del 10 al 18 i HARD del 19 al 27.

Quan l'usuari cliqui un dels botons, l'Activity crearà l'Intent per enviar-li el nivell seleccionat a l'Activity **Statistics** i l'iniciarà. Aquesta Activity obtindrà el nivell de l'Intent i preguntarà a la classe **WebServiceClient** si hi ha connexió a Internet. En cas negatiu, mostrarà un missatge d'error a l'usuari d'error de connexió; i en cas afirmatiu, cridarà el mètode *selectLevelStats* per obtenir les estadístiques i mostrar-les.

3.2.10 Cas d'ús: Consultar estadístiques d'usuari

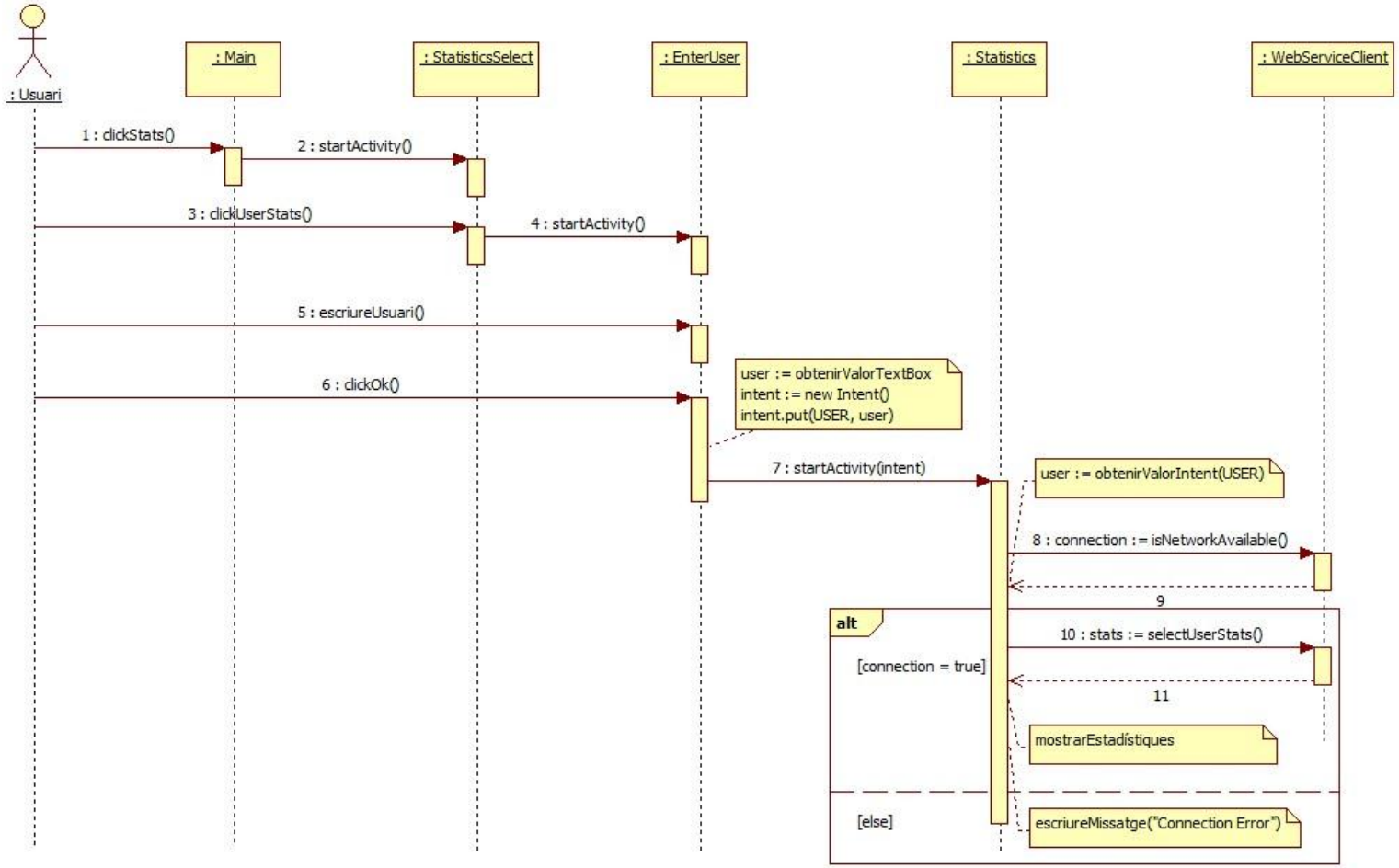


Figura 3.6. Diagrama de seqüència del cas d'ús: Consultar estadístiques d'usuari.

L'usuari clica el botó de stats, fet que provoca que la Activity principal **Main** comenci l'Activity **StatisticsSelect**. Aquí, l'usuari clicarà el botó d'estadístiques d'usuari i s'iniciarà l'Activity **EnterUser**.

Aquí l'usuari podrà introduir el nom de l'usuari del que vol consultar les estadístiques d'usuari. Aquesta informació s'encapsularà en un Intent i s'iniciarà l'Activity **Statistics**.

Aquesta Activity obtindrà el nom de l'usuari de l'Intent i preguntarà a la classe **WebServiceClient** si hi ha connexió a Internet. En cas negatiu, mostrarà un missatge d'error a l'usuari d'error de connexió; i en cas afirmatiu, cridarà el mètode *selectUserStats* per obtenir les estadístiques i mostrar-les.

3.2.11 Cas d'ús: Activar / Desactivar so

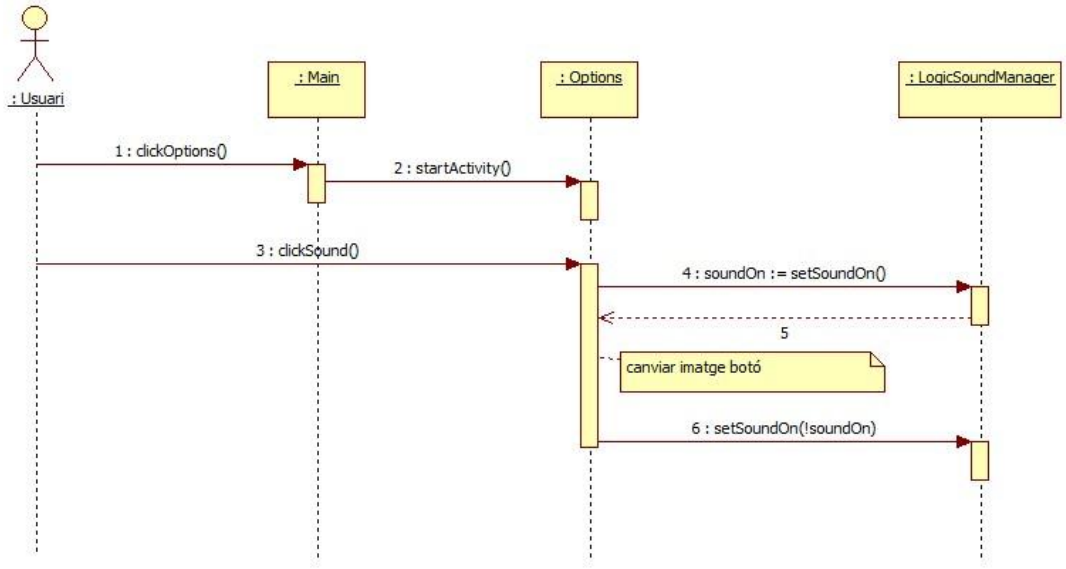


Figura 3.7. Diagrama de seqüència del cas d'ús: Activar/Desactivar so.

L'usuari clica al botó opcions de la pantalla principal del joc i aquest comença l'Activity **Options**.

Quan l'usuari clica el botó de so, aquest demana al **LogicSoundManager** l'estat del so. Si el so esta apagat, el botó canvia el text i es posa a ON, i si està encès, a OFF.

Finalment, s'actualitza l'estat del so cridant al mètode corresponent de la classe **LogicSoundManager**.

3.2.12 Cas d'ús: *Resetejar* nivells

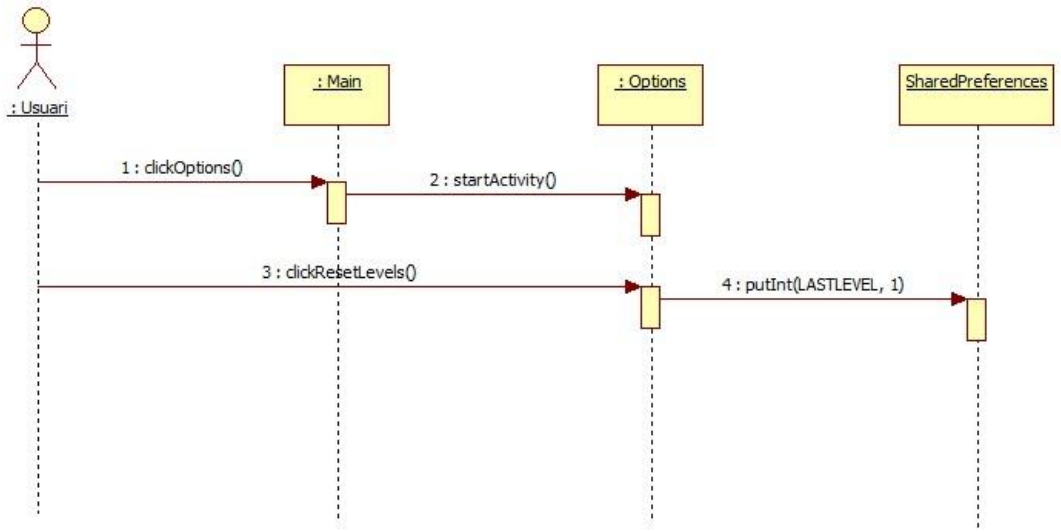


Figura 3.8. Diagrama de seqüència del cas d'ús: *Resetejar* nivells.

L'usuari clica al botó d'opcions de la pantalla principal del joc i aquest comença l'Activity **Options**.

Llavors l'usuari clica sobre el botó de *resetejar* nivells. Aquest botó es comunica amb la classe **SharedPreferences** i posa l'indicador de l'últim nivell a 1.

3.2.13 Cas d'ús: *Resetejar* usuari

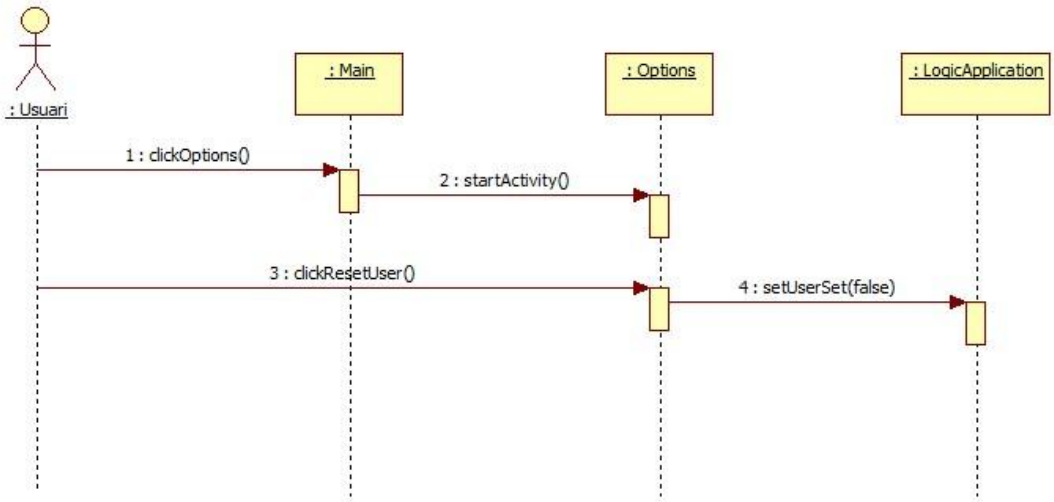


Figura 3.9. Diagrama de seqüència del cas d'ús: *Resetejar* usuari.

L'usuari clica al botó d'opcions de la pantalla principal del joc i aquest comença l'Activity **Options**.

Llavors l'usuari clica sobre el botó de *resetejar* usuari. Aquest botó es comunica amb la classe **LogicApplication** i li indica que posi la variable d'usuari com a no inicialitzada, fent que a partir d'ara es torni a preguntar per l'usuari al començar un nivell.

3.2.14 Cas d'ús: Consultar crèdits

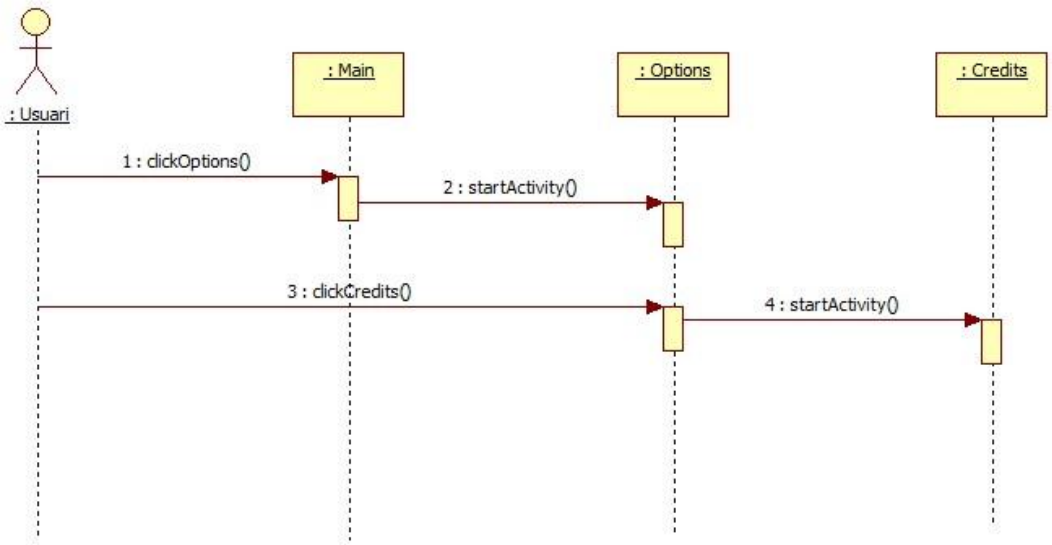


Figura 3.10. Diagrama de seqüència del cas d'ús: Consultar crèdits.

L'usuari clica al botó d'opcions de la pantalla principal del joc i aquest comença l'Activity **Options**.

Llavors l'usuari clica sobre el botó de crèdits. S'inicia l'Activity de **Crèdits** que no té cap interacció amb l'usuari (excepte el botó d'enrere), i mostra la informació de l'autor i el projecte de 3D Logic.

3.3 Paquet OpenGL

A continuació descriurem la "part OpenGL" de l'aplicació, dedicada exclusivament a mostrar el nivell de joc. Aquesta part inclou no només la visualització i crides OpenGL, sinó també l'estructura del cub (cub, cares i quadrats), els algorismes de selecció i els de comprovació de la solució.

3.3.1 Diferències d'OpenGL ES respecte OpenGL

Abans de començar amb el disseny de la part OpenGL de l'aplicació, hem de tenir en compte totes les diferències d'OpenGL ES respecte OpenGL que ens afectaran en el disseny de l'aplicació:

- **No existeix la semàntica de crides glBegin-glEnd.** En canvi, tenim les crides glDrawArrays i glDrawElements que ens permeten pintar a partir d'arrays i buffers de vèrtexs i índexs a vèrtexs.

- **No existeix el mode GL_QUADS**, que ens seria molt útil per pintar els quadrats. Per pintar cada quadrat, hem de pintar 2 triangles.

- **No existeix la crida glPolygonMode: només existeix el mode GL_FILL**, que omple el polígon. Per tant, haurem de pintar les arestes del cub (o sigui, les arestes dels quadrats del cub) pintant línies fent servir el mode GL_LINES.

- **No existeix mode SELECT**. L'implementador ha de crear els seus propis mètodes per seleccionar.

En 3D Logic farem servir la funció glDrawElements per pintar. El funcionament és el següent:

- Cridarem el mètode **glEnableClientState**(GL_VERTEX_ARRAY). Això és necessari per activar certes funcionalitats de client, que per defecte estan desactivades. A l'activar *GL_VERTEX_ARRAY*, activarem el vector de vèrtexs per ser utilitzat en les crides *glDrawElements*.

- Cridarem el mètode **glVertexPointer**, per apuntar al vector de vèrtexs, de la següent manera:

  ```
  glVertexPointer(int size, int type, int stride, Buffer pointer)
  ```
  on:
  - **size:** Nombre de coordenades per vèrtex. En el nostre cas, 3.
  - **type:** Tipus de dada de vèrtex. En el nostre cas, float.
  - **stride:** No ens afecta en 3D Logic, mantenim el valor 0 per defecte.
  - **pointer:** El Buffer creat a partir del vector de vèrtexs que associem.

- Finalment, farem ús de la crida **glDrawElements**:

```
glDrawElements (int mode, int count, int type, Buffer indices)
```

on:

- **mode:** El mode en que pintarem. Per exemple, si volem pintar triangles pintarem en *GL_TRIANGLES*.
- **count:** Nombre d'índexs a vèrtexs a pintar.
- **type:** Tipus de dada dels índexs. En el nostre cas, short.
- **indices:** Buffer creat a partir del vector d'índexs que associem.

3.3.2 Model conceptual (UML)

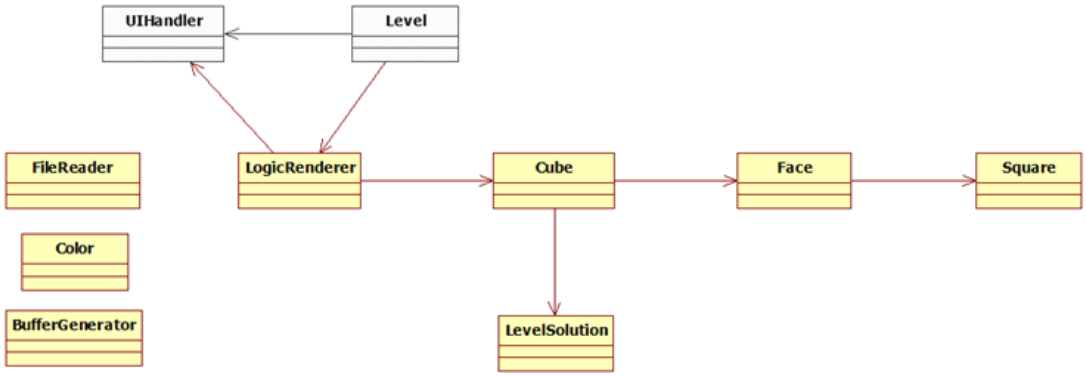


Figura 3.11. Diagrama UML amb les interaccions entre les classes que formen el paquet OpenGL.

El primer que salta a la vista d'aquest model conceptual son les classes UIHandler i Level, pintades d'un color diferent. És per indicar que aquestes classes, tot i formar part del paquet Android, també tenen rellevància en la part OpenGL.

Per part de la classe **UIHandler**, com hem explicat anteriorment, ens permetrà **comunicar el thread de pintat OpenGL amb el thread UI d'Android**.

En el cas de la classe **Level, serà la encarregada de gestionar la rotació i selecció que apliqui l'usuari**, mitjançant el mètode *onTouchEvent*, que és cridat cada cop que es toca la pantalla, i que ens retorna al thread UI d'execució.

Per la resta de classes, hem de destacar la importància de la classe **LogicRenderer, que ens realitzarà el pintat OpenGL**; les classes **Cube, Face i Square que representen la lògica de representació del cub**; i la classe **LevelSolution, que contindrà l'algorisme que calcula la solució** segons l'estat.

Les classes FileReader, Color i BufferGenerator son estàtiques. **FileReader ens permetrà llegir i tractar la informació dels arxius del cub**, generant-ne arrays i posant les dades necessàries en la representació Cube+Face+Square; **Color ens proporcionarà els colors** de cada solució parcial en arrays en format RGB, i **BufferGenerator és una classe auxiliar per a transformar arrays en buffers**, necessaris per les crides de pintat.

3.3.3 Representació del cub: informació base

Per representar el cub necessitem de certa informació base que és la mateixa per a tots els nivells del joc. Aquesta informació la guardarem en arxius de text que seran llegits per la classe FileReader i se n'obtindrà la informació desitjada.

Per explicar els formats dels arxius de representació del cub parlaré només de la primera cara, la cara frontal. Per la resta de cares es procedeix de la mateixa manera. Adjuntaré en peu de pàgina els noms dels arxius per si es volen consultar en el codi font de l'aplicació.

També és important notar que, per convenció, l'ordre de les cares és el següent: frontal, superior, esquerre, inferior, dreta i posterior. Els índexs dels vèrtexs del cub i dels quadrats interns es defineixen en aquest ordre, així com és l'ordre en que s'indiquen els quadrats bloquejats (veure secció 3.3.4).

Vèrtexs

Els vèrtexs del cub son tots els vèrtexs dels quadrats que formen les cares del cub. Això fa un total, eliminant repeticions, de 56 vèrtexs.

El centre del cub està situat a l'origen de coordenades.

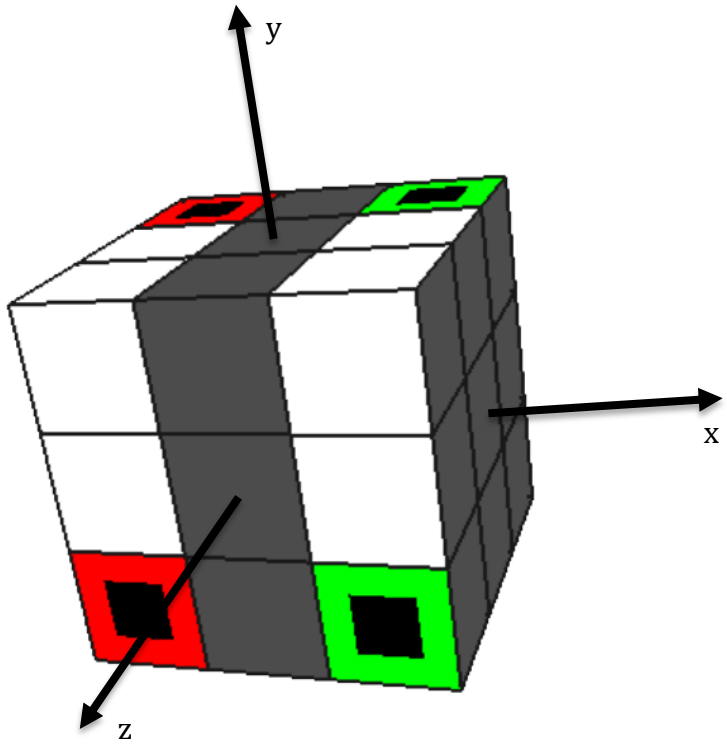


Figura 3.12. Representació del cub en l'espai.

Cada quadrat del cub serà de una unitat[14]. Per tant, l'arxiu de vèrtexs[15] tindrà la següent forma:

```
56 (nombre de vèrtexs)
-1.5 -1.5 1.5
-0.5 -1.5 1.5
0.5 -1.5 1.5
1.5 -1.5 1.5
-1.5 -0.5 1.5
-0.5 -0.5 1.5
0.5 -0.5 1.5
1.5 -0.5 1.5
-1.5 0.5 1.5
-0.5 0.5 1.5
0.5 0.5 1.5
1.5 0.5 1.5
-1.5 1.5 1.5
-0.5 1.5 1.5
0.5 1.5 1.5
1.5 1.5 1.5
...
```

Índexs

Com hem comentat anteriorment, en OpenGL ES no es poden pintar quadrats. Per tant, haurem de pintar 2 triangles x 9 quadrats per cara x 6 cares que fan un total de 108 triangles.

Per a fer-ho fem servir la funció *glDrawElements* d'OpenGL que **pinta a partir d'un buffer de vèrtexs** (obtingut a partir de l'arxiu de vèrtexs de l'apartat anterior) **i d'un buffer d'indexs a vèrtexs**. Els índexs a vèrtexs apunten als vèrtexs: l'índex 0 correspondrà al primer vèrtex definit, l'índex 1 al segon vèrtex definit, i així successivament.

Per la cara frontal del cub tenim aquesta distribució d'indexs a vèrtexs:

[14] Per facilitar la col·locació espacial dels vèrtexs del cub.
[15] Arxiu a res/raw/cube_vertices

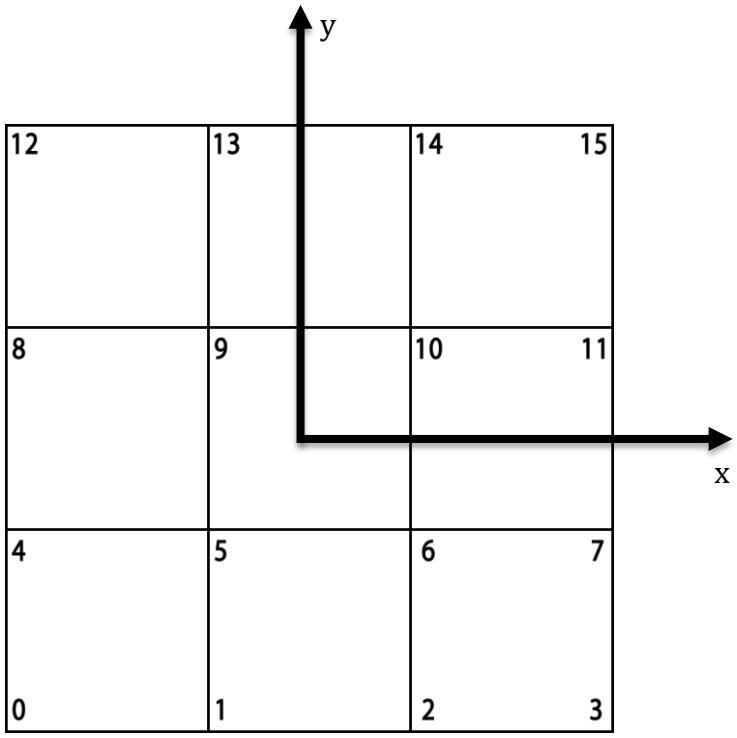


Figura 3.13. Representació de la cara frontal del cub amb els índexs a vèrtexs.

Com en OpenGL, **l'ordre dels vèrtexs importa**. Per tant, hem de tenir en compte que la normal del polígon dibuixat té la cara visible seguint l'ordre antihorari dels seus vèrtexs. I a més a més, per a cada quadrat hem de pintar dos triangles.

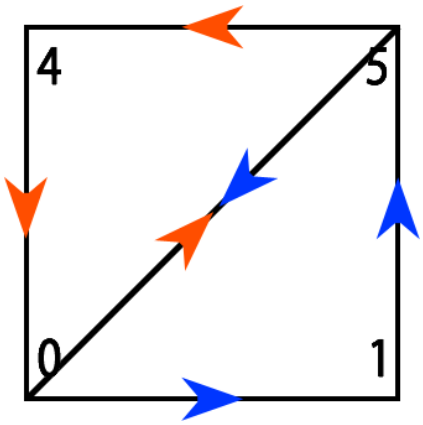


Figura 3.14. Representació de l'ordre en que s'han d'indexar els vèrtexs per pintar correctament els dos triangles que formen el primer quadrat de la primera cara del cub.

Per tant, l'arxiu d'índexs a vèrtexs del cub[16] queda definit de la següent manera:

```
108 (nombre d'índexs a vèrtexs)
// FRONT
0 1 5
5 4 0
1 2 6
6 5 1
2 3 7
7 6 2
4 5 9
9 8 4
5 6 10
10 9 5
6 7 11
11 10 6
8 9 13
13 12 8
9 10 14
14 13 9
10 11 15
15 14 10
...
```

Arestes

Per pintar les arestes del cub, pintarem línies amb el mode GL_LINES i la crida glDrawElements. Per tant, com que reaprofitarem el mateix arxiu de vèrtexs definit anteriorment, només ens farà falta definir un arxiu d'índexs a vèrtexs del cub per a arestes[17], que serà de la següent forma:

```
36 (nombre d'arestes)
0 3
4 7
8 11
12 15
0 12
1 13
2 14
3 15
...
```

[16] Arxiu a res/raw/cube_indices
[17] Arxiu a res/raw/cube_edge_indices

Quadrats interiors

L'última informació necessària per pintar el cub son els quadrats interiors[18]. Aquests quadrats ens serviran per poder diferenciar entre quadrats pintats:

- Si és un quadrat inicial i està seleccionat per l'usuari, es pintarà blanc.
- Si és un quadrat inicial i la seva solució parcial és correcta, es pintarà d'un color similar al seu però més fosc.
- Si és un quadrat inicial, no està seleccionat i la solució parcial no es correcta, el quadrat interior es pintarà negre.
- Si no es un quadrat inicial, no es pintarà res.

Els quadrats inicials es situaran a l'espai de la següent manera:

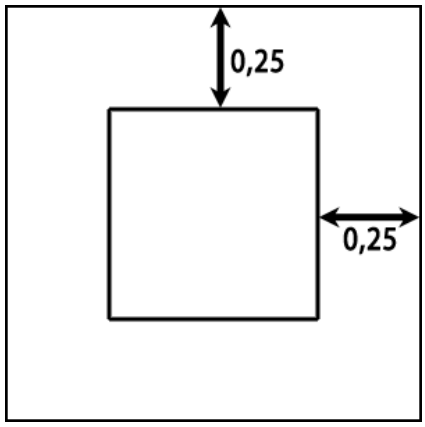


Figura 3.15. Representació en l'espai dels vèrtexs dels quadrats interiors.

Haurem de crear, doncs, un nou arxiu de vèrtexs per tots els vèrtexs dels quadrats interiors[19] :

```
216 (nombre de vèrtexs interiors)
-1.25 -1.25 1.5
-0.75 -1.25 1.5
-1.25 -0.75 1.5
-0.75 -0.75 1.5
-0.25 -1.25 1.5
0.25 -1.25 1.5
-0.25 -0.75 1.5
0.25 -0.75 1.5
0.75 -1.25 1.5
1.25 -1.25 1.5
0.75 -0.75 1.5
1.25 -0.75 1.5
-1.25 -0.25 1.5
```

[18] En aquesta secció explicarem com es defineixen els quadrats, i correspondrà a la següent secció, d'informació de nivell, explicar quins d'aquests quadrats inicials es pintaran, degut a que depèn del nivell.

[19] Arxiu a res/raw/cube_inner_vertices

```
-0.75 -0.25 1.5
-1.25 0.25 1.5
-0.75 0.25 1.5
-0.25 -0.25 1.5
0.25 -0.25 1.5
-0.25 0.25 1.5
0.25 0.25 1.5
0.75 -0.25 1.5
1.25 -0.25 1.5
0.75 0.25 1.5
1.25 0.25 1.5
-1.25 0.75 1.5
-0.75 0.75 1.5
-1.25 1.25 1.5
-0.75 1.25 1.5
-0.25 0.75 1.5
0.25 0.75 1.5
-0.25 1.25 1.5
0.25 1.25 1.5
0.75 0.75 1.5
1.25 0.75 1.5
0.75 1.25 1.5
1.25 1.25 1.5
...
```

3.3.4 Representació del cub: informació de nivell

Per representar les característiques del cub en cada nivell de joc necessitem de certa informació, emmagatzemada en un arxiu per a cada nivell[20], que tenen el següent contingut:

- **Nombre de solucions parcials.**

- **Les solucions parcials:** una línia per solució parcial i cada una s'escriu de la forma:
  - $CARA_{INICIAL}$ $QUADRAT_{INICIAL}$ $CARA_{FINAL}$ $QUADRAT_{FINAL}$.

- **Els quadrats bloquejats:** cada línia correspon als quadrats bloquejats d'una cara, amb cares ordenades segons l'ordre de cares vist a l'apartat 3.3.3:
  - N $QUADRAT_1$ $QUADRAT_2$ ... $QUADRAT_N$.
  - S'utilitza la paraula clau ALL si es vol indicar que tots els quadrats de la cara son bloquejats.

```
2
FRONT 0 UP 6
FRONT 2 UP 8
3 1 4 7
3 1 4 7
```

[20] Arxius a assets/levelX, on X és el número de nivell

```
ALL
ALL
ALL
ALL
```

Aquesta informació és tractada per la classe FileReader i no només en genera els vectors que ens serviran per pintar amb la crida glDrawElements, sinó que també n'omple la informació de cada quadrat a la classe Square i introdueix els quadrats inicials a la classe LevelSolution.

3.3.5 Integració Android amb OpenGL

Fins ara hem parlat de com està creat el cub a l'espai, i de com s'agafen i es distribueixen les dades per tal de ser representat el cub segons el nivell. Però, no hem parlat encara de com representarem l'escena, com pintarem els objectes i com seleccionarem.

Tot això ho comentarem en les següents seccions, però abans, és necessari fer una aturada i preguntar-nos: com s'integra Android amb OpenGL ES? Com passarem de les classes Activity a les classes amb codi OpenGL?

Ho farem amb la classe GLSurfaceView. Aquesta estén de la classe SurfaceView, que proporciona una superfície dedicada de dibuix incrustada dins la jerarquia de Views, i ens permet, entre d'altres coses:

- Pintar OpenGL en aquesta Surface.
- Definir-li Renderers propis que faran el pintat segons com desitgem.
- Pintar en un thread dedicat i independent del UI Thread.
- Situar aquest element GLSurfaceView dins d'un layout XML com ho faríem amb qualsevol altre View.

Finalment, només ens falta indicar a l'Activity en qüestió que farem servir un Renderer propi. Això es fa obtenint la GLSurfaceView definida al XML i utilitzant el seu mètode setRenderer:

```
public void  setRenderer(GLSurfaceView.Renderer renderer)
```

3.3.6 LogicRenderer

GLSurfaceView.Renderer és la interfície genèrica de pintat. LogicRenderer és la nostra implementació d'aquesta interfície, que a apart de les inicialitzacions i demés funcions que li definim, haurà de sobreescriure els 3 mètodes onSurfaceCreated, onSurfaceChanged i onDrawFrame.

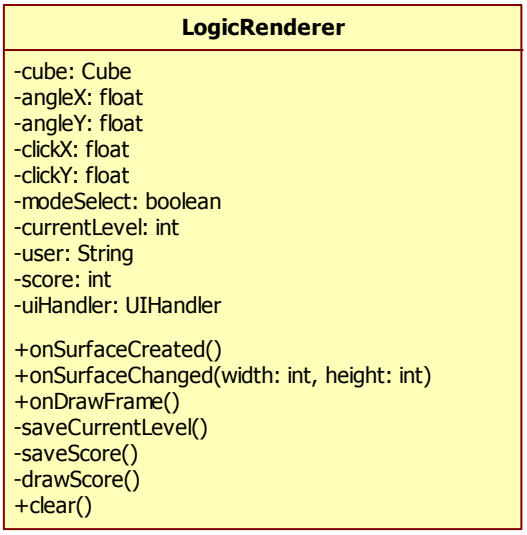


Figura 3.16. Classe LogicRenderer amb els atributs i mètodes més rellevants.

`onSurfaceCreated`

Mètode cridat quan la Surface és creada. És un bon lloc per fer les inicialitzacions que no canvien durant el procés de *renderitzat*, com per exemple el color de fons.

`onSurfaceChanged`

Mètode cridat quan la Surface ha canviat de mida, per exemple si l'aplicació permet orientació automàtica (vertical i horitzontal segons l'ús del dispositiu), es cridaria aquesta funció al girar el dispositiu mòbil.

En 3D Logic això no passa, ja que sempre està orientat verticalment, així que realment la Surface mai canvia de mida. Però aprofitarem aquest mètode igualment ja que ens proporciona els paràmetres amplada i altura necessaris per inicialitzar la matriu de projecció amb la crida *gluPerspective*.

```
// gluPerspective(fovy, aspect ratio, zNear, zFar)
gluPerspective(45.0f, width / height, 0.1f, 100.0f)
```

La càmera també la definirem en aquest mètode utilitzant la crida *gluLookAt*. En 3D Logic, estarà situada 10 unitats a distància en l'eix z, mirant el centre del cub i amb el up en la direcció positiva de l'eix y:

```
// gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ,
// upX, upY, upZ)
gluLookAt(gl, 0, 0, 10, 0, 0, 0, 0, 1, 0);
```

I finalment, carregarem la matriu identitat a la matriu MODELVIEW amb la crida *glLoadIdentity*.

onDrawFrame

Mètode encarregat de pintar el *frame* actual.

Aquest mètode també serà l'encarregat de fer les crides OpenGL per la rotació i de controlar l'estat del cub. Si estem en mode pintat, cridarem a la funció draw de la classe Cube, mentre que si estem en mode selecció cridarem la funció select de la classe Cube. Si l'usuari ha trobat la solució, guardarem la informació necessària i inicialitzarem el següent nivell.

A continuació descriurem un codi simplificat del mètode amb les característiques més importants:

```
Acció onDrawFrame
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
  glLoadIdentity()
  glRotatef(angleX, 0, 1, 0)
  glRotatef(angleY, 1, 0, 0)

  Si cube.isFinished llavors
    saveScore
    saveCurrentLevel
    initLevel
  Sino Si modeSelect llavors
    cube.select(clickX, clickY)
    modeSelect := false
  fSi

  cube.draw
  drawScore
fAcció
```

3.3.7 Cube

Fins ara hem explicat com es situa a l'espai el cub, amb tota la informació necessària de vèrtexs i arestes, com s'integra OpenGL ES en Android, on es situa l'observador i com s'inicialitzen les matrius de modelview i projecció.

Ara ens toca explicar l'estructura de classes per guardar la informació necessària per realitzar el pintat, selecció i els càlculs per trobar la solució. Això ho farem mitjançant les classes Cube, Face i Square. Comencem amb la classe Cube:

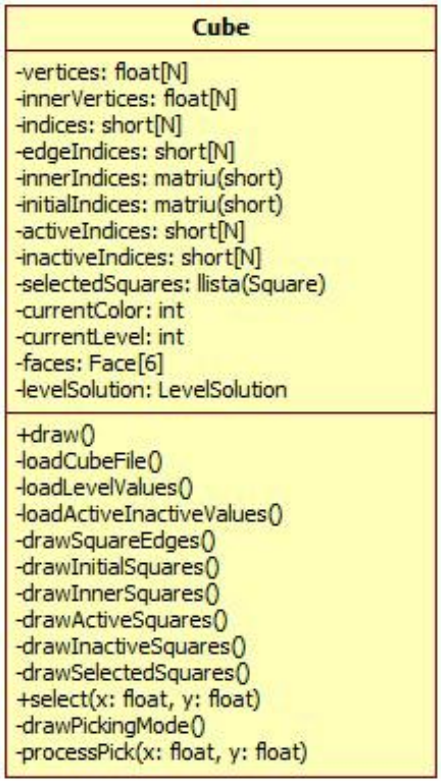


Figura 3.17. Classe Cube amb els atributs i mètodes més rellevants.

Atributs

La classe té com atributs tots els vectors necessaris per guardar la informació de vèrtexs i índexs. També té l'objecte levelSolution per càlculs de solució, la relació amb la classe Face i dos enters, currentColor (indica el color actual en que estem) i currentLevel (el nivell actual).

A més a més hi ha una variable de tipus Buffer[21] per cada vector de vèrtexs i índexs, necessaris per les crides a *glDrawElements*.

Inicialització del cub

Al constructor s'inicialitzaran les variables i es cridaran les funcions **loadCubeFile**, **loadLevelValues** i **loadActiveInactiveValues**, que carregaran els vectors i buffers de índexs i vèrtexs mitjançant la informació dels arxius del cub. També es faran les inicialitzacions a levelSolution dels quadrats inicials.

Pintat

El mètode públic **draw** serà cridat per **onDrawFrame** de LogicRenderer i pintarà tota la informació en pantalla utilitzant els següents mètodes privats:

[21] FloatBuffer o ShortBuffer segons si el vector associat és de shorts o de floats.

```
Acció drawSquareEdges // Per pintar les arestes
  glColor(0.1, 0.1, 0.1, alpha[22])
  glLineWidth(2)
  glPolygonOffset(1, 1)
  glEnable(GL_POLYGON_OFFSET_FILL)

  glEnableClientState(GL_VERTEX_ARRAY)
  glVertexPointer(3, GL_FLOAT, 0, vertexBuffer)
  glDrawElements(GL_LINES, edgeIndices.length,
  GL_UNSIGNED_SHORT, edgeIndexBuffer)

  glDisable(GL_POLYGON_OFFSET_FILL)
  glDisableClientState(GL_VERTEX_ARRAY)
fAcció
```

La crida *glPolygonOffset* juntament amb l'activació de *GL_POLYGON_OFFSET_FILL* ens permet definir la profunditat dels elements que pintem: es visualitzaran primer els objectes pintats amb un valor inferior.

```
Acció drawInitialSquares // Per pintar els quadrats inicials
  For i=1 to initialIndices.length do
    color = Color.getColor(i)
    glColor(color[0], color[1], color[2], alpha)
    glPolygonOffset(3, 1)
    glEnable(GL_POLYGON_OFFSET_FILL)

    glEnableClientState(GL_VERTEX_ARRAY)
    glVertexPointer(3, GL_FLOAT, 0, vertexBuffer)
    glDrawElements(GL_TRIANGLES, initialIndices[i].length,
    GL_UNSIGNED_SHORT, initialIndicesBuffer[i])

    glDisable(GL_POLYGON_OFFSET_FILL)
    glDisableClientState(GL_VERTEX_ARRAY)
  fFor
fAcció
```

La matriu initialIndices es de la forma: *initialIndices[NUM_SOLUCIONS_PARCIALS][2],* ja que hi ha 2 quadrats inicials per solució parcial. Per tant simplement recorrem aquesta matriu i gràcies a la classe Color pintem els quadrats inicials de cada solució parcial amb el seu color.

---

[22] El paràmetre alpha és sempre 1 excepte en els moments en que el cub es completa, que es fa un efecte fade in / fade out (és a dir, aparèixer i desaparèixer), on el paràmetre alpha varia per donar l'efecte de transparència. Aquest efecte no està explicat a la memòria perquè no és dels més importants però es pot consultar el seu comportament en el codi font en les funcions fadeIn i fadeOut de la classe Cube.

```
Acció drawInnerSquares
  // Per pintar els quadrats interiors inicials
  For i=1 to innerIndices.length do
    Si currentColor = i llavors glColor(1, 1, 1, alpha)
    Sino Si levelSolution.isWayCorrect(i) llavors
      color = Color.getInnerColor(i)
      glColor(color[0], color[1], color[2], alpha)
    Sino glColor(0, 0, 0, alpha)
    fSi

    glPolygonOffset(2, 1)
    glEnable(GL_POLYGON_OFFSET_FILL)

    glEnableClientState(GL_VERTEX_ARRAY)
    glVertexPointer(3, GL_FLOAT, 0, innerVertexBuffer)
    glDrawElements(GL_TRIANGLES, innerIndices[i].length,
    GL_UNSIGNED_SHORT, innerIndicesBuffer[i])

    glDisable(GL_POLYGON_OFFSET_FILL)
    glDisableClientState(GL_VERTEX_ARRAY)
  fFor
fAcció
```

Per pintar els quadrats interiors inicials procedim igual que quan hem pintat els quadrats inicials però tenim en compte l'estat per triar el color, tal i com hem comentat a l'apartat 3.3.3.

```
Acció drawActiveSquares
  // Per pintar els quadrats actius i no seleccionats
  glColor(1, 1, 1, alpha)
  glPolygonOffset(4, 1)
  glEnable(GL_POLYGON_OFFSET_FILL)

  glEnableClientState(GL_VERTEX_ARRAY)
  glVertexPointer(3, GL_FLOAT, 0, vertexBuffer)
  glDrawElements(GL_TRIANGLES, activeIndices.length,
  GL_UNSIGNED_SHORT, activeIndicesBuffer)

  glDisable(GL_POLYGON_OFFSET_FILL)
  glDisableClientState(GL_VERTEX_ARRAY)
fAcció
```

Poc a comentar en quan els quadrats actius, ja que segueixen l'estructura ja explicada anteriorment. Pels quadrats inactius no adjuntarem pseudocodi, ja que és com pintar els actius però canviant el color (a 0.3 per pintar gris) i els índexs (inactiveIndices i inactiveIndicesBuffer).

```
Acció drawSelectedSquares
  // Per pintar els quadrats seleccionats
  ForEach square in selectedSquares do
    color = Color.getColor(square.getSelectedColor)
    selectedIndices = square.getIndices
```

```
        selectedIndicesBuffer =
        BufferGenerator.generateShortBuffer(selectedIndices)[23]

        glColor(color[0], color[1], color[2], alpha)
        glPolygonOffset(3, 1)
        glEnable(GL_POLYGON_OFFSET_FILL)

        glEnableClientState(GL_VERTEX_ARRAY)
        glVertexPointer(3, GL_FLOAT, 0, vertexBuffer)
        glDrawElements(GL_TRIANGLES, selectedIndices.length,
        GL_UNSIGNED_SHORT, selectedIndicesBuffer)

        glDisable(GL_POLYGON_OFFSET_FILL)
        glDisableClientState(GL_VERTEX_ARRAY)
    fForEach
fAcció
```

Per a pintar els quadrats seleccionats, recorrem l'estructura de selectedSquares, obtenint-ne el color i els índexs i pintant cada quadrat de forma individual.

Selecció

Veure secció 3.3.10.

3.3.8 Face

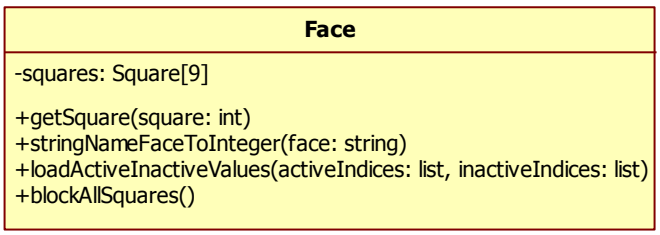


Figura 3.18. Classe Face amb els atributs i mètodes més rellevants.

La classe Face ens proporcionarà les següents funcionalitats:

- **getSquare:** Per obtenir el quadrat desitjat.
- **stringNameFaceToInteger:** Per convertir l'string de cara en l'enter corresponent. És a dir, la cara amb string "FRONT" és la primera cara (segons l'ordre establert de cares), per tant és la cara 0; la cara "UP" és la cara 1, etc. En cas de que l'string corresponent no correspongui a cap cara, retornarà -1.

[23] Es podria millorar l'eficiència d'aquesta funció fent que el ShortBuffer fos un atribut de la classe Square i així no haver-lo de recalcular per cada crida draw.

- **loadActiveInactiveValues:** Funció auxiliar per construir les llistes de quadrats actius i inactius.

- **blockAllSquares:** Bloqueja tots els quadrats de la cara. Útil per quan ens trobem la paraula "ALL" al llegir els valors bloquejats de nivell.

3.3.9 Square

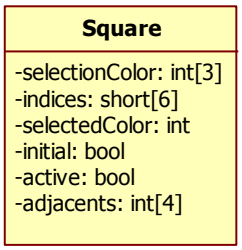


Figura 3.19. Classe Square amb els atributs i mètodes més rellevants.

Finalment tenim la classe Square que ens persistirà informació sobre els quadrats. No té cap mètode públic excepte els getters/setters dels atributs següents:

- **selectionColor:** Vector amb el color associat al quadrat en l'algorisme de selecció per color. NO confondré amb el color que l'usuari ha seleccionat.

- **indices:** Vector amb els índexs a vèrtex del quadrat. Ens servirà per quan pitnem els quadrats seleccionats.

- **selectedColor:** Enter que determina el color (la solució parcial) seleccionat per l'usuari per aquest quadrat.

- **initial:** Boolèa que indica si el quadrat és inicial o no.

- **active:** Booleà que indica si el quadrat està actiu o no.

- **adjacents:** Vector d'enters per indicar quins son els quadrats adjacents d'aquest quadrat.

3.3.10 Level: Rotació i selecció

El mètode onTouchEvent d'una Activity d'Android és cridat quan un event de tocar la pantalla succeeix. Com que Level és la classe Android que defineix la vista d'OpenGL, farem servir aquest mètode per detectar clicks que faci l'usuari per rotar i seleccionar.

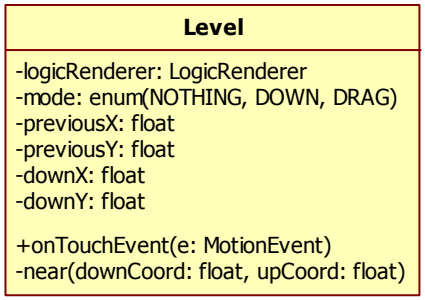


Figura 3.20. Classe Cube amb els atributs i mètodes més rellevants.

El mètode onTouchEvent el definirem de la següent forma:

```
Funció onTouchEvent (e: MotionEvent) Retorna: bool
  x := e.getX
  y := e.getY

  Selecciona e.getAction

    Cas: ACCIÓ_DOWN // Toquem la pantalla
      downX := x
      downY := y
      mode := DOWN
    fCas

    Cas: ACCIÓ_UP // Deixem de tocar la pantalla
      Si mode = DOWN llavors
        logicRenderer.setModeSelect
        logicRenderer.setClickX(x)
        logicRenderer.setClickY(y)
      fSi
      mode := NOTHING
    fCas

    Cas: ACCIÓ_MOVE // Arrastrem el dit per pantalla
      Si mode != NOTHING llavors
        logicRenderer.addAngleX(x - previousX)
        logicRenderer.addAngleY(y - previousY)
        Si !near(downX, x) o !near(downY, y) llavors
          mode := DRAG
        fSi
      fSi
    fCas

  fSelecciona

  previousX := x
  previousY := y

  Retorna true
fFunció
```

Quan toquem la pantalla, guardem la posició en que s'ha tocat en les variables down i posem l'estat DOWN.

Quan deixem de tocar la pantalla hem de mirar en quin estat estem.
- Si estem en estat DOWN vol dir que hem seleccionat un quadrat, per tant avisem a LogicRenderer de que estem en mode selecció i li donem les coordenades X i Y del click. Finalment, ens posem en estat NOTHING.
- Si estem en estat DRAG, estem rotant el cub i per tant no hem de seleccionar cap quadrat.

Quan arrastrem el dit per pantalla, rotem el cub. Per això li donem a logicRenderer els factors x i y de rotació. A més a més, hem d'utilitzar la funció near per saber si estem a prop o lluny del click original:

```
Funció near(downCoord: float, upCoord: float) Retorna: bool
  Retorna Abs(downCoord - upCoord) < DIST[24]
fFuncio
```

El motiu d'aquesta comprovació es que un clic en pantalla no és perfecte, és a dir, l'usuari pot voler clicar un quadrat però moure el dit uns píxels abans de deixar-lo anar. Tampoc podem fer que el cas ACCIÓ_UP sempre seleccionés, ja que l'usuari podria només voler rotar, i al deixar anar el dit si ho fes sobre un quadrat el pintaria.

Per tant amb aquesta comprovació mirarem si l'usuari ha arrastrat prou el dit com per considerar-se que estem en mode rotació.

3.3.11 Algorisme de selecció

Com hem dit anteriorment, OpenGL ES no ofereix, a diferència de OpenGL, un mecanisme de selecció. Per tant, és el programador l'encarregat d'implementar els seus propis mètodes per seleccionar.

D'entre els diferents mecanismes de selecció coneguts s'ha triat el de **selecció per codificació de color (Color Coding o Color Picking),** degut a que és un mètode de selecció relativament senzill d'implementar i, si bé hi ha altres mètodes com el "Ray Casting" que son més eficients, també son força més complexos d'implementar.

Aquest mètode consisteix en el següent: **es repinta l'escena amb tots els objectes pintats en un color en concret**, i es guarda la informació dels colors de cada objecte en alguna estructura. Llavors, **s'agafa el color en pantalla del píxel seleccionat per l'usuari** i es busca quin és **l'objecte pintat en aquest color. Ja tenim l'objecte seleccionat.**

---

[24] DIST és la distància en que considerem que ja estem en un cas de rotació i no de selecció. En 3DLogic està configurat a 20 píxels.

Tot aquest procés es realitza de forma transparent a l'usuari, per tant un cop sabem quin quadrat s'ha seleccionat, es pinta (si s'escau) i es torna a pintar l'escena normalment.

A 3D Logic, pintarem cada quadrat del cub d'un color diferent, això vol dir que els 2 triangles que conformen cada quadrat seran pintats amb el mateix color, i que haurem de fer servir 108/2 = 54 triangles. A més a més, distribuirem els colors segons les cares de la manera següent[25], sent la última component la que varia entre el primer quadrat de la cara i l'últim:

	Primer color RGB	Últim color RGB
Frontal	0, 0, 28	0, 0, 255
Superior	127, 0, 28	127, 0, 255
Esquerra	255, 0, 28	255, 0, 255
Inferior	0, 255, 28	0, 255, 255
Dreta	127, 255, 28	127, 255, 255
Posterior	255, 255, 28	255, 255, 255

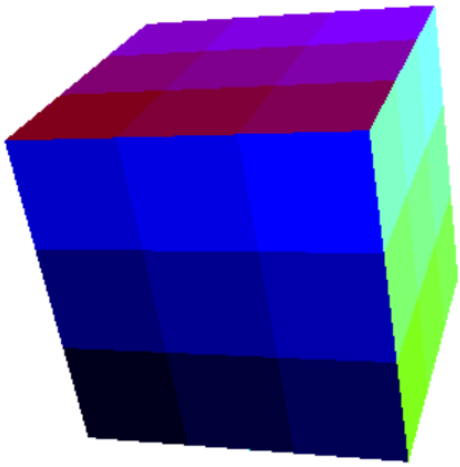


Figura 3.21. Representació del pintat del cub en mode selecció.

[25] La distribució de colors podria ser diferent, però l'he triat així per poder descartar cares segons les components.

3.3.12 Algorisme de comprovació de solució

Cada cop que l'usuari pinta un quadrat, hem de comprovar no només si l'usuari ha trobat la solució del nivell, sinó també si ha activat/desactivat una solució parcial[26] del nivell.

Per fer aquesta comprovació tenim una classe dedicada: LevelSolution.

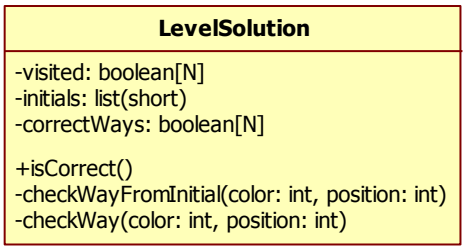


Figura 3.22. Classe LevelSolution amb els atributs i mètodes més rellevants.

Aquesta té informació de:

- **Quadrats visitats:** Vector de booleans per marcar els quadrats ja visitats durant la comprovació de la solució.

- **Quadrats inicials:** Llista dels quadrats inicials (només un de la parella de cada color). Aquesta llista s'omple durant la càrrega de la informació de nivell.

- **Camins correctes:** Vector de booleans indicant quines solucions parcials son correctes.

LevelSolution disposa d'un mètode públic *isCorrect* que retorna un booleà indicant si s'ha trobat la solució o no. Aquesta funció fa les inicialitzacions convenients i crida el mètode *checkWayFromInitial* per cada un dels quadrats inicials del nivell de joc:

[26] Entenem per solució parcial de nivell (o també anomenat camí o solució de color) un camí correcte entre dos quadrats inicials del mateix color, encara que aquest camí no sigui el correcte per a la solució global del nivell.

```
Funció isCorrect Retorna: bool
  For i=1 to size(initials)
    Si comprovarCamiDesdeInicial(i, initials[i]) = true
    llavors caminsCorrectes[i] = true
    Sino
      caminsCorrectes[i] = false
      nivellCorrecte = false
    fSi
  fFor

  Retorna nivellCorrecte
fFunció
```

La funció *checkWayFromInitial* agafa tots els adjacents del quadrat i busca el camí correcte cridant la funció *checkWay*.

```
Funció checkWayFromInitial (color: int, position: int)
Retorna: bool
  quadrat = getSquare(position)
  adjacents = quadrat.getAdjacents

  ForEach adjacent € adjacents do
    Si comprovaCami(color, adjacent) = true llavors
      Retorna true
    fSi
  fForEach

  Retorna false
fFunció
```

La funció *checkWay* segueix un esquema Backtracking, per tal d'expandir un arbre amb tots els camins possibles i anar podant segons les següents condicions:

- Si el quadrat és visitat, estaríem mirant una branca ja visitada, per tant retornem false.
- Si el quadrat no és actiu no forma part del camí, per tant podem i retornem false.
- Si el color del quadrat és diferent al color del camí, no formarà part del camí. Podem i retornem false.
- Si el quadrat es inicial i el color del quadrat és el del camí, forma part del camí i és el final del camí, per tant retornem true.

Si no es compleix cap de les condicions anteriors, es fa la crida recursiva sobre els adjacents del quadrat on estem per seguir expandint l'arbre de solucions i trobar o no el camí. I si alguna d'aquestes crides retorna true, ja no fa falta que seguim buscant: hem trobat un camí correcte.

A continuació adjuntem el pseudocodi de la funció *checkWay:*

```
Funció checkWay (color: int, position: int)
Retorna: bool
  Si visitat[position] llavors Retorna false

  visitat[position] := true
  quadrat = getSquare(position)

  Si noActiu(quadrat) llavors Retorna false
  Si color(quadrat) != color llavors Retorna false
  Si esInicial(quadrat) llavors Retorna true

  adjacents = quadrat.getAdjacents

  ForEach adjacent € adjacents do
    Si comprovaCami(color, adjacent) llavors Retorna true
  fForEach

  Retorna false
fFunció
```

3.4 Paquet WebService

Finalment descriurem la "part WebService" de l'aplicació mòbil, dedicada exclusivament a comunicar-se amb el servidor web. En aquesta part s'explica la lògica que hi ha darrere de la consulta d'estadístiques de nivell, d'usuari, el *parseig* de la informació i la inserció de puntuacions en el sistema.

3.4.1 Model conceptual (UML)

Figura 3.23. Diagrama UML del paquet OpenGL.

Per consultar i inserir estadístiques al servidor només en fan falta aquestes dos classes, WebServiceClient, encarregada de fer les crides al servei web, i LogicSaxHandler, encarregada de *parsejar* la informació que retorna el servei web.

3.4.2 WebServiceClient

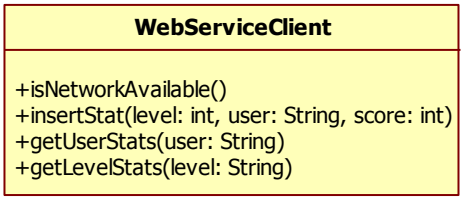


Figura 3.24. Diagrama UML del paquet OpenGL.

La classe WebServiceClient és una classe senzilla que permetrà a les classes Android i OpenGL obtenir o inserir la informació que desitgi al servidor. Els mètodes que ofereix son:

- **isNetworkAvailable:** Comprova si el dispositiu mòbil té connexió a Internet. Si no en té, no farem cap crida a servidor i retornarem un missatge d'error de connexió a l'usuari.

- **insertStat:** Podrem inserir una estadística al servidor amb nivell *level*, usuari *user* i puntuació *score*.

- **getUserStats:** Aquest mètode ens retornarà una matriu de strings amb les millors estadístiques, sobre l'usuari *user*. Aquesta matriu de strings tindrà la informació del nivell i la puntuació per cada resultat, amb un màxim de

10 resultats. Si l'usuari no existeix o hi hagués un error de connexió es tornaria *null* i ho haurem de tractar.

- **getLevelStats:** Mètode que retornarà les millors puntuacions estadístiques del nivell *level* en una matriu de strings. Aquesta matriu tindrà un màxim de 10 resultats i informarà sobre l'usuari i la puntuació de cada estadística. Si hi ha un error de connexió durant la consulta, es tornarà *null* i s'haurà de tractar.

3.4.3 LogicSAXHandler

El servei web ens torna la informació de les estadístiques en format XML. Aquí entra en joc LogicSAXHandler, que estén de SAXHandler, una classe dedicada al parseig d'XMLs.

Aquesta classe té mètodes que detecten el començament i final d'etiquetes XML, permetent recuperar-ne el valors desitjats. Per tant, la classe WebServiceClient s'ajudarà de LogicSAXHandler per obtenir la informació de les estadístiques de nivell i d'usuari en un format que li sigui adequat per retornar-ho a les classes Android que hauran de mostrar aquesta informació en pantalla[27].

[27] No entro en més detall respecte aquesta classe ja que considero que és simplement un *parsejador* de XML i seria entrar en detalls d'implementació que poden ser consultats en el codi font.

4 Mapes navegacionals

A continuació, s'exposaran els mapes navegacionals de l'aplicació. Ens servirà per tenir una idea general de com estan distribuïts els menús del joc, quins son i què fan els botons i com interactuen les pantalles entre sí.

4.1 Jugar partida

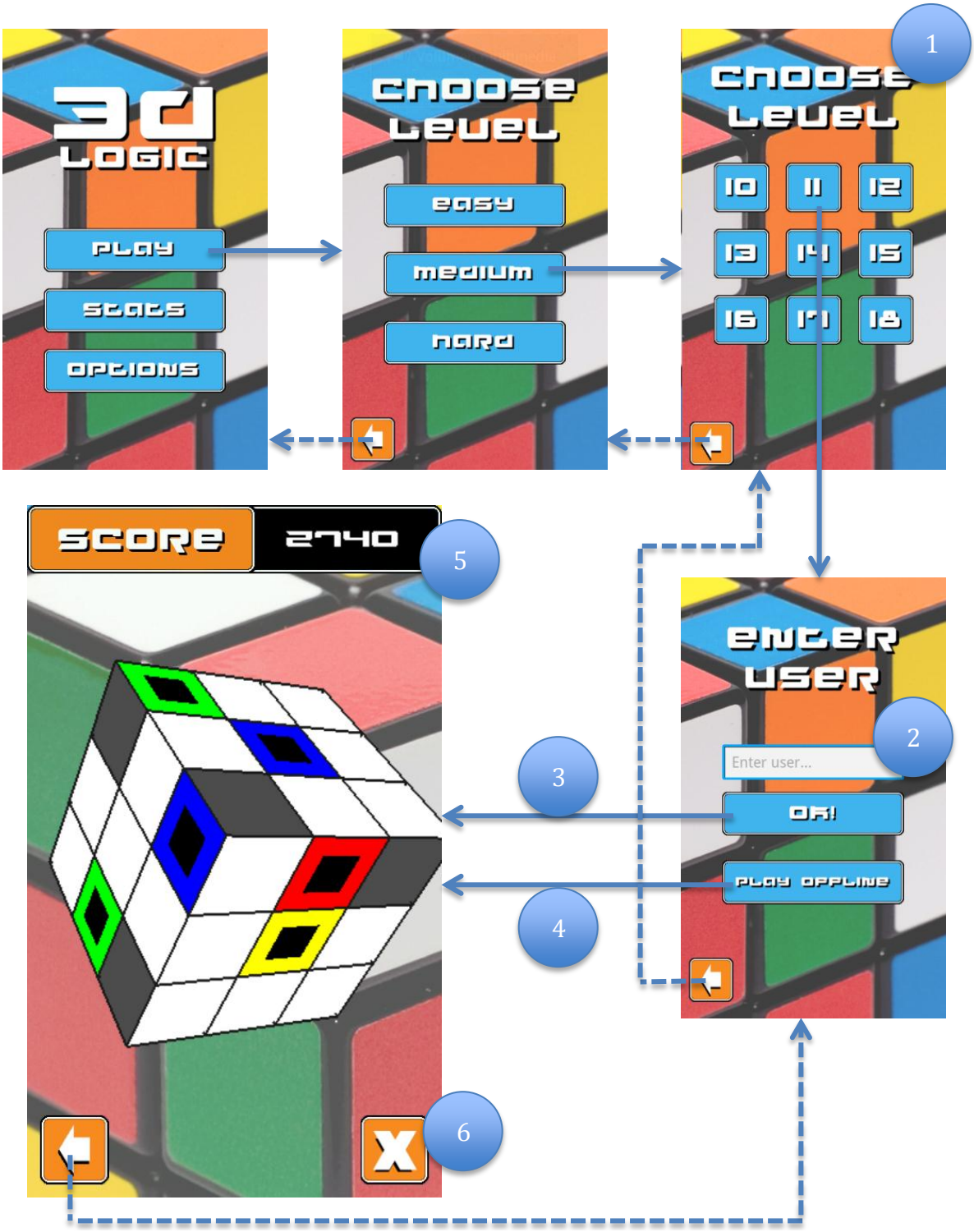


Figura 4.1. Mapa navegacional del cas d'ús: Jugar partida.

Llegenda:

1. La pantalla de selecció de nivell mostra els nivells segons la dificultat. És a dir, si a la pantalla anterior haguéssim tocat a EASY, els nivells mostrats serien del 1-9; i si haguéssim tocat HARD, del 19 al 27. A més a més, es mostraran actius només aquells nivells que l'usuari ja hagi completat i el primer sense completar.

2. Caixa de text per introduir l'usuari per jugar online.

3. Si cliquem OK, jugarem online amb l'usuari introduït a la caixa de text. Si no introduïm usuari, l'aplicació mostra un missatge avisant a l'usuari i no s'executa el nivell.

4. Si cliquem PLAY OFFLINE, no es guardaran les puntuacions del joc al sistema d'estadístiques. Per tant no es fa cas del valor introduït a la caixa de text.

5. El marcador amb la puntuació del joc. Sempre apareix, encara que l'usuari no jugui online[28].

6. Botó CLEAR o botó de reset. Serveix perquè tots els quadrats pintats per l'usuari siguin esborrats i es torni a l'estat inicial (excepte el marcador, que es manté igual) del nivell.

[28] Vaig creure convenient que es veiés sempre, ja que la idea es que els usuaris juguin online (sempre que disposin de connexió), i per tant que puguin veure el marcador es un element per motivar-los a que ho facin.

4.2 Consultar estadístiques

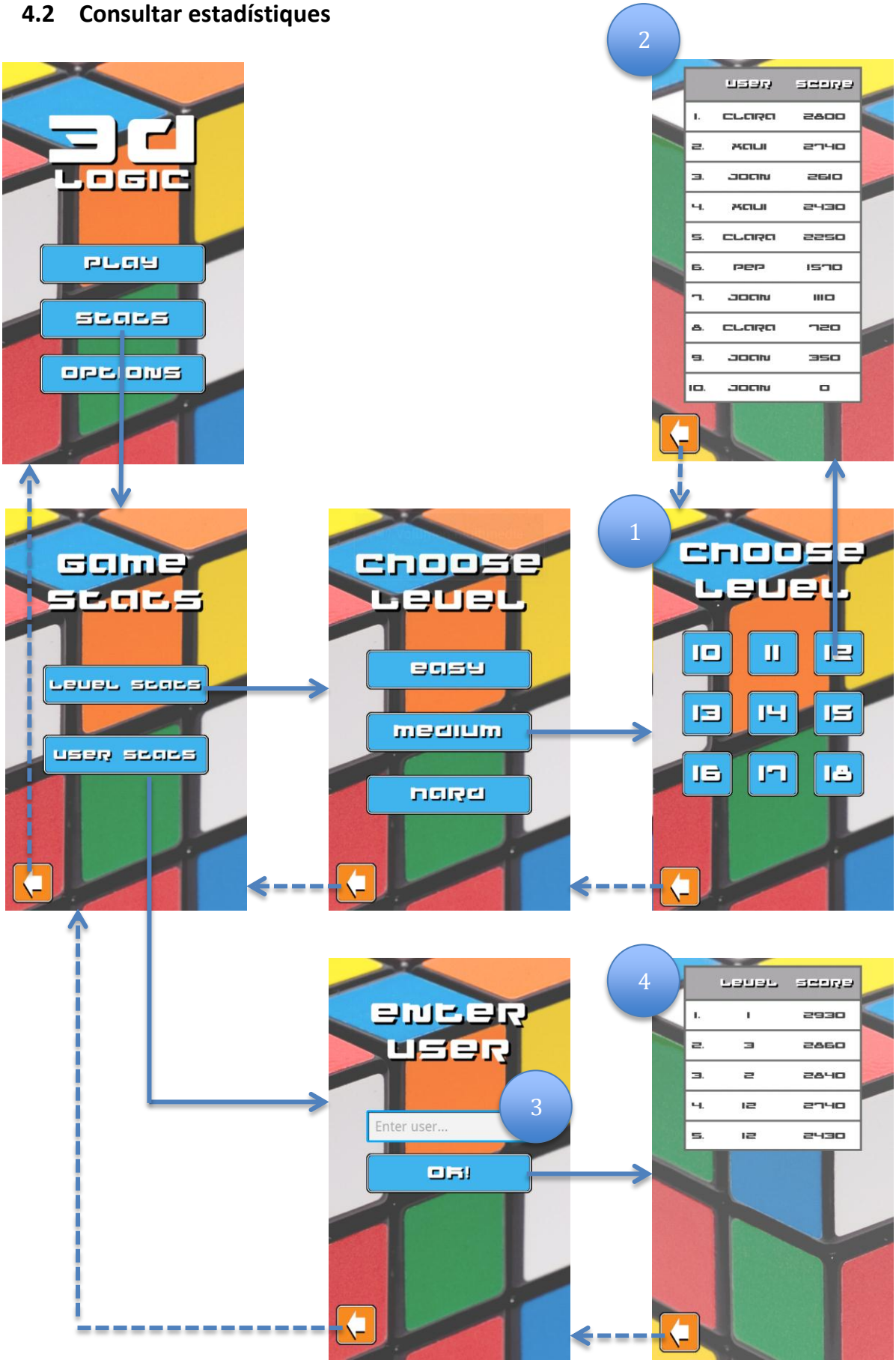


Figura 4.2. Mapa navegacional del cas d'ús: Consultar estadístiques.

Llegenda:

1. La pantalla de selecció de nivell mostra els nivells segons la dificultat. És a dir, si a la pantalla anterior haguéssim tocat a EASY, els nivells mostrats serien del 1-9; i si haguéssim tocat HARD, del 19 al 27.

2. Es mostren com a màxim les 10 millors estadístiques del nivell triat, i es mostra la següent informació: posició, usuari i puntuació.

3. Caixa de text per introduir l'usuari per jugar online.

4. Es mostren com a màxim les 10 millors estadístiques de l'usuari escrit en la caixa de text. Per cada estadística es mostra: posició, nivell i puntuació.

4.3 Pantalla opcions

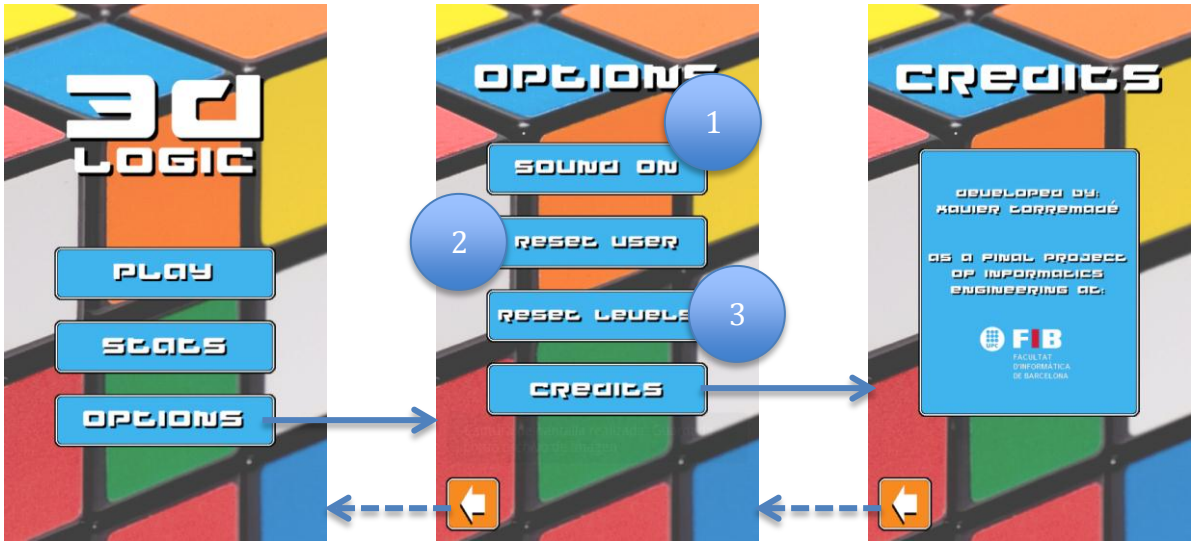


Figura 4.3. Mapa navegacional del cas d'ús: Consultar estadístiques.

Llegenda:

1. El botó serveix per desactivar o activar el so. El text del botó s'actualitza, és a dir, quan el so està desactivat mostra SOUND OFF i quan el so està activat mostra SOUND ON.

2. Botó per *resetejar* l'usuari. Si es clica, la següent partida que es faci es preguntarà a l'usuari pel seu nom.

3. El botó per *resetejar* els nivells. Prement-lo, es desactivaran tots els nivells que l'usuari hagi completat excepte el primer.

5 Planificació

5.1 Diagrama de Gantt

A continuació adjunto el diagrama de Gantt amb la planificació que s'ha seguit pel projecte:

Nombre de tarea	Duración	Comienzo	Fin
Estudi previ	**9 días**	**lun 05/09/11**	**mar 13/09/11**
Repàs OpenGL	4 días	lun 05/09/11	jue 08/09/11
Estudi OpenGL ES	5 días	vie 09/09/11	mar 13/09/11
Anàlisis	5 días	mié 14/09/11	dom 18/09/11
Disseny	**71 días**	**lun 19/09/11**	**dom 11/12/11**
Paquet android	7 días	lun 19/09/11	dom 25/09/11
Paquet opengl	10 días	lun 26/09/11	mié 05/10/11
Paquet webservice	2 días	sáb 10/12/11	dom 11/12/11
Implementació	**64 días**	**jue 06/10/11**	**mié 21/12/11**
Configuració escena	7 días	jue 06/10/11	vie 14/10/11
Configuració cub	15 días	sáb 15/10/11	sáb 29/10/11
Selecció	7 días	dom 30/10/11	sáb 05/11/11
Creació nivells	2 días	dom 06/11/11	lun 07/11/11
Pantalles android	7 días	mar 08/11/11	jue 24/11/11
Quadrats inicials	4 días	vie 25/11/11	mar 29/11/11
Control nivells actius	2 días	mié 30/11/11	jue 01/12/11
So	2 días	vie 02/12/11	sáb 03/12/11
Botó borrar	2 días	dom 04/12/11	lun 05/12/11
Puntuació	4 días	mar 06/12/11	vie 09/12/11
Estadístiques	6 días	lun 12/12/11	sáb 17/12/11
Algorisme compr. Solu	4 días	dom 18/12/11	mié 21/12/11
Tests	5 días	jue 22/12/11	lun 26/12/11
Documentació	49 días	jue 06/10/11	jue 12/01/12

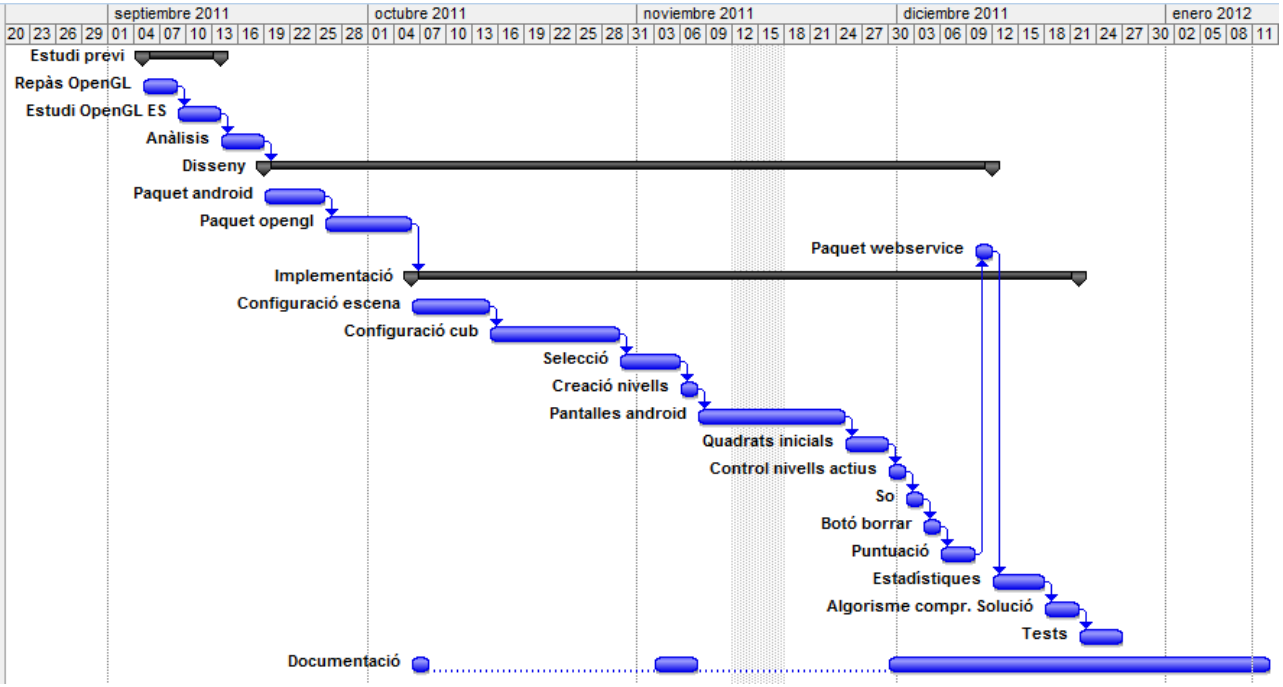


Figura 5.1. Diagrama de Gantt

Per la planificació comentada anteriorment cal tenir en compte que **el projecte s'ha fet mentre treballo a l'empresa on estic actualment,** concretament, treballant 5 hores cada dia. Per tant, tots els **dies laborables es compten les hores de la tarda** i no el dia complert**.**

També queda clar amb el Gantt que **els caps de setmana també m'han servit per fer projecte**, excepte algunes excepcions com per exemple el cap de setmana de cap d'any.

En el Gantt es pot observar uns dies en gris, degut a que del 10 al 19 de novembre, per motius familiars, no vaig poder dur a terme res del projecte. Aquest fet ha provocat un retràs en el projecte, i per poder-lo entregar durant aquest semestre, s'han hagut de fer bastantes més hores del previst durant el mes de desembre.

El projecte s'ha realitzat de forma seqüencial excepte el paquet WebService. Això és degut a que tota la part servidor i gestió d'estadístiques es va proposar durant la realització del projecte (confirmant-se durant la realització de l'informe de desenvolupament del projecte). És per aquest motiu que **després de la tasca de puntuació, s'ha fet el disseny del paquet WebService de l'aplicació i finalment s'ha implementat la funcionalitat**.

Tota **la implementació del servidor s'ha realitzat durant la tasca d'Estadístiques**, juntament amb la implementació de la part client de l'aplicació mòbil.

Finalment, per la **realització de la memòria**, hi ha tingut 3 fases:

- La primera correspon a realitzar els diagrames i les explicacions necessàries per explicar el disseny en la seva primera fase.

- La segona fase correspon als canvis que he hagut de fer al disseny d'aspectes que no s'havien previst o tingut en compte, així com canvis que he vist durant la implementació.

- Finalment, la tercera fase i la més llarga ha consistit en acabar la realització de la memòria, des dels apartats que no s'havien començat (introducció, conclusions, millores...), refinar i acabar els apartats d'anàlisis i disseny, format i índexs del document, i finalment fer una segona iteració pel document per revisar faltes i explicacions que no hagin quedat clares.

5.2 Descripció de les tasques

A continuació descriuré les 6 tasques principals i comentaré també les desviacions que hi ha hagut respecte a les previsions inicials:

5.2.1 Estudi previ

- **Temps estimat:** 15 dies.
- **Temps real:** 9 dies.
- **Descripció:** Les diferències entre OpenGL i OpenGL ES a alt nivell són fàcils d'entendre, per tant em va portar menys temps del previst. Els problemes han sigut més en la implementació, com veurem després.

5.2.2 Anàlisis

- **Temps estimat:** 5 dies.
- **Temps real:** 5 dies.
- **Descripció:** Cap incidència durant la realització de l'anàlisi. S'hagués pogut fer en menys temps si s'hagués descartat abans la possibilitat de fer el projecte en OpenGL ES 2.0.

5.2.3 Disseny

- **Temps estimat:** 14 dies.
- **Temps real:** 19 dies.
- **Descripció:** La desviació ve principalment per la integració d'Android amb OpenGL, ja que no es veu clar en els manuals d'Android Developers com comunicar els dos threads.

  A més a més, es va plantejar la opció d'utilitzar textures per a pintar els quadrats, en comptes de fer-ho amb colors, però es va descartar perquè augmentava bastant la complexitat del projecte i l'única avantatge era pel pintat dels quadrats interiors, que es podia fer simplement canviant la textura.

  Finalment, hem d'afegir el disseny del WebService de client que no estava previst en la planificació inicial.

5.2.4 Implementació

- **Temps estimat:** 50 dies.
- **Temps real:** 62 dies.
- **Descripció:** La desviació de la implementació ve marcada sobretot per la selecció. Tot i que l'algorisme de selecció per color és senzill i fàcil

d'entendre, la implementació ha donat problemes degut a les funcions pròpies d'OpenGL ES en Android, i va suposar una desviació important.

En aquesta desviació també hi entra la implementació de les classes WebService de client i el servidor i la seva base de dades.

En altres punts, però, s'ha pogut acabar abans. Un exemple és el desenvolupament dels quadrats inicials, que s'ha realitzat en 4 dies en compte dels 7 inicialment previstos.

5.2.5 Tests

- **Temps estimat:** 5 dies.
- **Temps real:** 5 dies.
- **Descripció:** Els tests s'han realitzat segons els terminis previstos.

5.2.6 Documentació

- **Temps estimat:** 40 dies.
- **Temps real:** 49 dies.
- **Descripció:** La desviació amb la documentació ve donada perquè s'han acabat fent més funcionalitats de les previstes inicialment, que han requerit ser documentades.

6 Possibles millores

En aquest capítol explicaré diferents millores que s'han pensat durant la realització del projecte però que han estat descartades perquè se sortien de la planificació d'hores destinades a un Projecte de Final de Carrera.

A més a més de presentar les millores també proposaré una possible solució per dur-la a terme[29], sempre intentant reaprofitar el màxim l'estructura i codi utilitzat per desenvolupar el projecte.

6.1 Generació de nivells per part de l'usuari

Una de les millores seria crear un sistema per a que els usuaris de 3D Logic poguessin crear els seus propis nivells de joc.

La millora es plantejaria de la següent manera: l'usuari podria accedir a una **pantalla per crear un nivell de joc**. Se li preguntaria a l'usuari quantes cares tindria disponibles el nivell (amb un mínim de 1 i un màxim de 6) i un cop l'usuari introduís aquesta informació es mostraria una pantalla similar a la que es mostra quan es juga un nivell però amb les següents diferències:

- El cub, inicialment, es mostraria tot en blanc.
- Es mostraria un botó per a que l'usuari pugui introduir la següent parella de colors inicials.
- Es mostraria un botó per a que l'usuari pugui introduir un quadrat bloquejat.
- Es mostraria un botó per esborrar de la última parella de colors inicials inserida.
- Es modificaria el botó de *Reset* per a esborrar tot el cub, inclosos els quadrats inicials.
- No es mostraria el marcador de puntuació.
- Es mostraria un botó per confirmar el nivell.

L'usuari afegiria parelles de colors prement el botó corresponent. A més a més, podria moure un color inicial prement sobre seu i a continuació prement un altre quadrat sense pintar. Tot això es podria fer aprofitant l'algorisme de selecció ja desenvolupat.

Al prémer el botó de confirmar nivell, s'executaria un algorisme semblant al de comprovació de resultat que comprovaria si la solució proposada per l'usuari pot ser resolta. En cas afirmatiu aquest nivell es guardaria, i en cas contrari s'avisaria a l'usuari de que la proposta de nivell és incorrecta.

---

[29] Consultar l'Annex per entendre els detalls sobre la base de dades i el servidor que seran comentats en aquest capítol.

Un cop s'hagués comprovat que el nivell és correcte, **el nivell es guardaria en la memòria del mòbil** en format de tipus text seguint el mateix format que els nivells estàndards del joc. Per tant l'aplicació recorreria tots els quadrats del cub per obtenir-ne la informació i crear l'arxiu corresponent, posant-li com a nom el nom de nivell que triés l'usuari i guardant-lo en la carpeta corresponent als nivells creats per l'usuari dins del seu dispositiu mòbil.

Evidentment, aquesta millora també comportaria la **creació d'una nova pantalla per a poder consultar els nivells creats per l'usuari**, que es classificarien pel nom.

6.2 Guardar nivells creats per l'usuari en servidor

Seguint el punt anterior, una altra millora seria que els nivells creats per l'usuari no es guardin en la memòria del mòbil sinó que es **guardin en el servidor**, fent així que tots els nivells creats fossin accessibles per tots els usuaris.

Per dur a terme aquesta millora, hauríem de modificar la base de dades creant una **nova taula anomenada NIVELLS** amb les següents columnes:

- **Nom:** El nom del nivell.
- **Autor:** Nom de l'usuari que l'ha creat.
- **Data:** La data de creació del nivell.
- **Info:** Informació del nivell seguint el format dels arxius de text de nivell. Possibles tipus: BLOB o TEXT.

També s'hauria d'**adequar el WebService** per tal de poder afegir un nou nivell creat per l'usuari i fer els canvis pertinents a les classes de client de webservice de l'aplicació mòbil.

6.3 Sistema de comptes d'usuari

Una altra millora possible per 3D Logic seria substituir l'actual sistema d'usuaris (que no controla la duplicitat) per un **sistema de comptes d'usuari**.

Per a fer-ho, es **crearia una pantalla de Login** a l'iniciar el nivell de joc on l'usuari posaria el seu nom d'usuari i contrasenya. En aquesta mateixa pantalla es permetria que l'usuari es creés la seva pròpia compta.

La lògica de guardar l'usuari a LogicApplication es canviaria, i ara es **guardaria l'usuari en memòria del mòbil**, per tal de que l'usuari no hagués de fer Login cada cop que executa l'aplicació[30]. A més a més, el botó "Reset User" es

[30] Degut a que fer Login en l'aplicació es incòmode en un dispositiu mòbil, la millor opció és recordar l'última sessió. D'aquesta forma funcionen la majoria d'aplicacions mòbil que requereixen iniciar sessió com Gmail, Gtalk, Facebook o Twitter.

substituiria per el **botó "Change User", per canviar d'usuari**, i s'obriria la mateixa pantalla de Login comentada anteriorment.

S'haurien de **modificar els mètodes del WebService** per permetre inserir un nou usuari a la base de dades. A més a més, aquesta es modificaria de la següent manera:

- Es crearia una **nova taula anomenada USUARI**, amb els següents camps:
  - **Username:** El nom de l'usuari.
  - **Password:** Contrasenya de l'usuari.
  - Altres camps com nom complert, data creació, localitat...
- Es modificaria la taula d'estadístiques, en concret la columna user, que passaria a ser una foregin key a la taula d'usuaris.
- Es modificaria la taula de nivells, canviant la columna autor que passaria a ser una foregin key a la taula d'usuaris.

6.4 Integració amb xarxes socials

Finalment, es podria aprofitar tota la lògica nova d'usuari comentada en l'apartat anterior per integrar-ho en xarxes socials, per tal de **publicar els teus millors resultats a Facebook, Twitter**, etc[31].

Per aconseguir-ho, es podria crear en l'aplicació mòbil una **nova pantalla de preferències d'usuari**. Aquí, es podrien configurar les diferents comptes de xarxes socials, i hi hauria un checkbox per permetre a l'usuari publicar els seus resultats a aquestes[32]. **Aquesta informació seria persistida a la taula d'usuaris en base de dades**.

[31] A més a més, seria recomanable crear pàgines d'aplicació a xarxes socials com Facebook. Avui en dia, aparèixer en les xarxes socials és la millor manera de publicitar un joc, sobretot si es publica gratuïtament. És per aquest motiu que seria una millora interessant per un joc com 3DLogic.

[32] Aprofitant les APIs de cada xarxa social.

7 Conclusions

7.1 Objectius complerts

Considero que s'han complert els objectius comentats a la introducció d'aquesta memòria.

S'ha realitzat una aplicació que permet jugar al joc 3D Logic, que ofereix una interfície gràfica correcta per l'usuari, amb certes característiques pròpies d'altres jocs per dispositius mòbils, com per exemple el bloqueig de nivells, i amb un sistema centralitzat d'estadístiques per incentivar la competició entre usuaris.

Els objectius marcats en l'apartat de possibles millores son molt interessants però se sortien de la planificació inicial per un projecte de final de carrera. Tot i així s'han comentat i serien interessants d'aplicar en un futur.

7.2 Valoració personal

Aquest projecte ha sigut per a mi la meva **primera iniciació real en el món del desenvolupament de jocs en 3D per dispositius mòbils**, i concretament en OpenGL ES i Android.

Degut a la meva afició pels videojocs i també pel mon dels *smartphones*, fer aquest projecte ha sigut una **gran oportunitat per mi per aprendre**. Fins ara les experiències més similars que havia tingut eren el desenvolupament d'aplicacions Android i certa iniciació en un *framework* anomenat Cocos2D en plataformes iOS.

El fet de tenir certa experiència en Cocos2D m'ha fet plantejar en més d'un moment si OpenGL ES és una bona plataforma per desenvolupar jocs. Com a mínim, i com a opinió totalment personal, **per a jocs senzills en 2D no recomanaria fer servir OpenGL ES**, degut a que *frameworks* com Cocos2D tenen conceptes de *sprites*, accions, efectes i transicions que faciliten molt el desenvolupament. I a més, **s'aconsegueixen normalment efectes millors que en OpenGL ES i, sobretot, en molt menys temps**.

És una bona idea desenvolupar jocs en 3D mitjançant OpenGL ES? Doncs depèn. És evident que *frameworks* ens faciliten la feina, però a vegades no ens deixen arribar al nivell de detall que voldríem, I tampoc hi ha dubte que OpenGL ES és i serà una molt bona API per a desenvolupar aplicacions en 3D. Dependrà més del coneixement del desenvolupador sobre OpenGL ES, del temps que es requereixi per desenvolupar-lo, i en definitiva, de les característiques del projecte.

El que no tinc cap dubte és que, per a qualsevol persona que desenvolupi jocs, ja sigui amb OpenGL ES o amb altres *frameworks*, conèixer aquesta API és una gran base. **I això és un dels grans valors que n'extrec de la realització d'aquest projecte**.

Com a punts negatius, haig de comentar que he notat per part d'Android Developers molta deixadesa cap a la integració amb OpenGL ES. La majoria d'explicacions i codis d'exemple que he trobat han sigut d'usuaris que han tingut els mateixos problemes que jo. En definitiva**, la referència d'Android Developers en tot el relacionat a GLSurfaceView i OpenGL ES és clarament insuficient**.

Un altre aspecte negatiu durant el desenvolupament del projecte ha sigut que, degut a les desviacions de la planificació del projecte i els problemes personals que em van obligar a perdre una setmana i mitja al novembre, se m'ha retardat el projecte. Això ha provocat que el desembre hagi hagut de dedicar-hi moltes hores i el desenvolupament no s'ha pogut fer amb la calma que un desitjaria. Tot i així, al final **el balanç és positiu, ja que s'han complert els objectius previstos**.

Però a més d'aportar-me coneixements pràctics sobre desenvolupament en OpenGL ES, també m'ha aportat molt més. Per exemple, he recordat conceptes d'anàlisis i disseny d'aplicacions, casos d'ús i notació en UML que feia bastant de temps que no ho feia servir. I com no, m'aporta una **gran satisfacció personal**, perquè ha suposat no només complir amb els objectius marcats amb el projecte, sinó també acabar la carrera en la que porto 5 anys estudiant.

En definitiva, estic molt satisfet per com ha anat el desenvolupament del projecte, per tot allò que m'aporta, tant personalment com professionalment, i per tancar una de les etapes més importants de la meva vida. Només falta desitjar que les noves etapes que estan per venir siguin iguals o millors.

7.3 Agraïments

A **Lluís Pérez Vidal**, pel seguiment del projecte, resolució de dubtes i consells sobre les decisions preses en el projecte.

Als **familiars i amics** que m'han donat suport i ànims durant la realització del projecte.

I finalment, a totes aquelles **fonts d'Internet**, des d'usuaris anònims a blocs dedicats a Android, que m'han servit d'ajuda per entendre les complicacions d'aquest projecte i que m'han proporcionat exemples útils per a poder-lo desenvolupar correctament.

8 Annex: Disseny servidor

En aquest annex explicarem a grans trets com hem creat el servidor i quins components el formen. Cal dir que aquesta solució triada és una solució amb els seus avantatges i inconvenients però en cap cas ha de ser la millor o la única solució al problema.

La tria de les tecnologies ha tingut en compte sobretot la senzillesa, ja que aquest projecte es centrava en l'aplicació mòbil i no es podia, per qüestió de temps, desenvolupar un servidor molt complex.

8.1 Arquitectura servidor

Figura 8.1. Arquitectura del servidor

Per la part servidor, disposem d'un ordinador[33] on hi hem instal·lat un servidor d'aplicacions GlassFish per atendre les peticions de les aplicacions mòbils, i d'una base de dades en MySQL per a guardar la informació relativa a les estadístiques.

8.2 Configuració servidor

Com hem vist, tenim com a servidor d'aplicacions GlassFish, que és lliure i desenvolupat per Sun Microsystems, actual Oracle. GlassFish està basat en el Sun Java System Application Server d'Oracle, que és un derivat d'Apache Tomcat, i utilitza un component anomenat Grizzly per a escalabilitat i velocitat.

[33] El meu ordinador personal de casa, amb un Windows 7. No és l'idoni per a fer de servidor però per aquest projecte és suficient.

A més a més farem servir Jersey, que és una implementació de JAX-RS per construir serveis web RESTFULL, i que proporciona una API que permet als desenvolupadors estendre Jersey per adaptar-lo a les seves necessitats.

Per tant amb Jersey podem utilitzar les anotacions de JAX-RS. El servei web de 3D Logic en farà servir i algunes de les més representatives son les següents:

- **@Path:** El valor d'aquesta anotació és la direcció URI indicant on estarà situada la classe.

- **@GET:** Per indicar que es processaran peticions GET.

- **@Produces:** Per indicar el tipus de representació que s'enviarà al client. Per exemple, "text_plain" retornarà text pla, "APPLICATION_XML" retornarà un arxiu en format XML...

- **@QueryParam:** Paràmetres enviats a la classe per ser tractats.

Per tant, un cop escrit el codi encarregat d'inicialitzar el servidor[34], ja podem centrar-nos en els *Resources*, que son els encarregats de rebre les peticions per part del client, i que utilitzen les anotacions comentades anteriorment.

A continuació descrivim, amb un pseudocodi molt simplificat, el comportament del nostre *Resource*:

```
Clase LogicWebService
@Path "/statistics"

  @GET
  @Produces "APPLICATION_XML"
  @Path "/query"
  Funció getStats (@QueryParam level: int, @QueryParam user:
  string)
  Retorna: Statistics[35]
    Si level != null llavors
       sql := construirSQLNivell(level)
    Sino sql := construirSQLNivell(level)
    fSi

    connexioBD := connectar_BD
    resultat := connexioBD.aplicarConsulta(sql)

    Retorna construirEstadistiques(resultat)
  fFunció
```

[34] Per més informació, consultar codi font de la classe Main de 3DLogicWebService.

[35] Classe auxiliar per a representar el XML d'estadístiques.

```
    @GET
    @Produces "text/plain"
    @Path "/insert"
    Funció insertStats (@QueryParam level: int, @QueryParam
    user: string, @QueryParam score: int)
    Retorna: String
      connexioBD := connectar_BD
      resultat := connexioBD.insertar(level, user, score)

      Retorna resultat
    fFunció
fClase
```

Per tant, seguint el codi anterior, l'aplicació mòbil crearà les següents URI per a accedir als serveis web:

- ***NOM_SERVIDOR*/statistics/query?level=*LEVEL***, per a consultar les estadístiques del nivell LEVEL.

- ***NOM_SERVIDOR*/statistics/query?user=*USER***, per a consultar les estadístiques de l'usuari USER.

- ***NOM_SERVIDOR*/statistics/insert?level=*LEVEL*&user=*USER*&score=*SCORE***, per a inserir una estadística del nivell *LEVEL*, usuari *USER* i puntuació *SCORE*.

8.3 Disseny base de dades

Per a 3D Logic tindrem una base de dades MySQL on guardar les estadístiques. Aquesta tindrà una taula anomenada *statistics* creada de la següent forma:

```
create table statistics (
  id MEDIUMINT NOT NULL AUTO_INCREMENT,
  level INT(3) NOT NULL,
  user CHAR(30) NOT NULL,
  score SMALLINT NOT NULL,
  PRIMARY KEY (id)
);
```

És cert que, com que la base de dades només disposa d'una taula i la informació a guardar és senzilla, no seria necessari utilitzar una base de dades i ho podríem guardar en un arxiu al servidor. Però he cregut convenient fer servir base de dades per dos motius:

- **Organització:** La base de dades ens permet organitzar molt bé la informació de les estadístiques, fent-ne fàcil les consultes i insercions, i tenint-ho centralitzat en una sola taula, al contrari que tenint arxius plans.

- **Canviabilitat:** Al capítol 6 Possibles millores hem parlat de modificacions de la base de dades per afegir comptes d'usuari i altra informació. Amb una base de dades els canvis serien relativament senzills: un script per la creació de les taules noves i un altre script pels canvis de les dades ja

existents. Amb arxius plans s'hauria de fer manualment o fer un programa que els *parsegés* i construís nous arxius, i seria més costós.

8.4 Publicació servidor

Es publica el servidor mitjançant un proveïdor de servei DNS, NO-IP[36]. Això ens permetrà que la classe WebServiceClient accedeixi al servei web mitjançant un nom i no la IP.

Aquest fet es de vital importància degut a que el nostre servidor està en una xarxa que té IP dinàmica, per tant s'hauria de canviar el codi de l'aplicació mòbil cada cop que canviés d'IP, cosa que és totalment inviable.

[36] http://www.no-ip.com/

9 Bibliografia

- Pàgina web oficial d'Android Developers.
  http://developer.android.com

- Pàgina web de Stackoverflow, per ajudes amb el desenvolupament i la implementació.
  http://stackoverflow.com/

- Pàgina web Droidnova, sobre desenvolupament en Android.
  http://www.droidnova.com/

- Sèrie de 6 tutorials de Per-Erik Bergman sobre Android i OpenGL ES 1.0, molt útils per iniciar-se.
  http://blog.jayway.com/2009/12/03/opengl-es-tutorial-for-android-part-i/

- All Music Library, pàgina web amb sons gratuïts utilitzats en 3D Logic:
  http://www.allmusiclibrary.com/free_sound_effects.php

- Pàgina web oficial d'Oracle, per a consultar referència de JAVA.
  http://www.oracle.com/technetwork/java/index.html

- Pàgina web oficial de Glassfish.
  http://glassfish.java.net/

- Pàgina web oficial de Jersey.
  http://jersey.java.net/

- Entrada de la pàgina Mkyong amb un exemple útil per entendre els paràmetres de JAX-RS.
  http://www.mkyong.com/webservices/jax-rs/jax-rs-queryparam-example/

- Pàgina web oficial de MySQL
  http://www.mysql.com/

- Pàgina web de NO-IP, el proveïdor de servei DNS.
  http://www.no-ip.com/

- Sèrie d'enllaços amb exemples pràctics per entendre les crides RESTfull en Android.
  http://lukencode.com/2010/04/27/calling-web-services-in-android-using-httpclient/
  http://senior.ceng.metu.edu.tr/2009/praeda/2009/01/11/a-simple-restful-client-at-android/
  http://timewasted.net/?p=127

- Article de Bare Bones Coder per a fer servir fonts personalitzades en elements Android:
  http://www.barebonescoder.com/2010/05/android-development-using-custom-fonts/

- Codehead's Bitmap Font Generator, extremadament útil per a pintar string textures. La classe TexFont del codi pertany a Karl Walsh:
  http://www.codehead.co.uk/cbfg/

- Wikipedia per informació de caràcter general i descripcions simples per iniciar el projecte.
  http://www.wikipedia.org/